Message Correlation for Conversation Reconstruction in Service Interaction Logs

Hamid R. Motahari Nezhad, Regis Saint-Paul, Boualem Benatallah

School of Computer Science and Engineering
The University of New South Wales
Sydney, Australia
{hamidm|regiss|boualem}@cse.unsw.edu.au

Fabio Casati, Periklis Andritsos

University of Trento
Trento, Italy
{casati|periklis}@dit.unitn.it

Technical Report
UNSW-CSE-TR-0709

March 2007

Abstract

The problem of understanding the behavior of business processes and of services is rapidly becoming a priority in medium and large companies. To this end, recently, analysis tools as well as variations of data mining techniques have been applied to process and service execution logs to perform OLAP-style analysis and to discover behavioral (process and protocol) models out of execution data. All these approaches are based on one key assumption: events describing executions and stored in process and service logs include identifiers that allow associating each event to the process or service execution they belong to (e.g., can correlate all events related to the processing of a certain purchase order or to the hiring of a given employee). In reality, however, such information rarely exists.

In this paper, we present a framework for discovering correlations among messages in service logs. We characterize the problem of message correlation and propose novel algorithms and techniques based on well-funded principles and heuristics on the characteristics of conversations and of message attributes that can act as identifier for such conversations. As we will show, there is no right or wrong way to correlate messages, and such correlation is necessarily subjective. To account for this subjectiveness, we propose an approach where algorithms suggest candidate correlators, provide measures that help users understand the implications of choosing a given correlators, and organize candidate correlators in such a way to facilitate visual exploration. The approach has been implemented and experimental results show its viability and scalability on large synthetic and real-world datasets. We believe that message correlation is a very important and challenging area of research that will witness many contributions in the near future due to the pressing industry needs for process and service execution analysis.

1 Introduction

The problem of understanding the behavior of information systems and the processes and services they support is rapidly becoming a priority in medium and large companies. This is demonstrated by the proliferation of tools for the analysis of process executions, service interactions, and service dependencies and by recent research work in process data warehousing and process discovery. Indeed, the adoption of business intelligence techniques for business process improvement is the primary concern for medium and large companies [7].

The opportunity for such analysis comes with the increased automated support, which corresponds to the ability to observe (collect events on) executions and interactions, thereby building information sources that can be used for the analysis. Typical applications consist in monitoring process and service executions, deriving statistics such as average process durations, or identifying steps in the process where the most time is spent (bottlenecks). Recently, variations of data mining techniques have also been applied to process and service execution logs to achieve important results such as (i) process and protocol discovery [20, 14] and (ii) root cause analysis for performance or quality degradations in process and service executions [18, 6].

All these approaches and solutions to business process and service analysis are based on one key assumption: events describing executions include an *instance ID* which identifies the set of events that belong to the same *execution* (also called *case*) of a process or service, e.g., belong to the processing of the same purchase order. Without this element that correlates events, execution analysis is difficult if not impossible. In such situations, we are neither able to compute basic statistics such as average process durations (e.g., we cannot find the average time elapsed between event A and B in a process because we cannot say which pairs A, B belong to the same process execution), nor to perform more complex tasks such as process and protocol discovery.

In reality, the situations in which these correlators are present are very limited, which means that the applicability of previous research (including the one by the authors in [14]) is also limited. In process execution, instance IDs are essentially present only for those processes supported by a workflow management system (WfMS), and specifically for that part of the business process supported by the WfMS. Today a small percentage of processes are supported by a WfMS and even in those cases only a small portion of the entire business process is WfMS-based. Hence, process analysis today is limited to the portions of the processes supported by the WfMS. Taken from the services side, the problem is similar: when studying *conversations* among services (a conversation is a sequence of message exchanges between parties for achieving a certain goal, e.g., completing a purchase order), we rarely find conversation identifiers in web service execution logs. Only if the ser-

vice implementation is tightly coupled with the logging infrastructure (e.g., if IBM WebSphere Integration Developer and IBM Process Server are used together [15]), then it is possible that identifiers are included. However, in many cases the logging infrastructure is offered as a standalone product (as in the case of HP SOA Manager and CA SOA Manager) that is not linked to individual service implementations. In these cases, the information about identifiers is not known to the logging infrastructure, but is hidden somewhere in the exchanged messages. In fact, this is one of the main obstacles that the authors faced on integrating automated process/protocol discovery solutions in HP SOA Manager, which is a monitoring and management infrastructure for Web services, and was one of the motivations of this work.

This paper tackles the problem of discovering correlations among messages exchanged by services. Specifically, our goal is that of correlating and grouping messages based on the conversation they belong to. This is equivalent to deriving some *conversation ID*s to label each message. In particular, we define the problem of message correlation, we present algorithms for correlation, and we show how to perform semi-automated correlation by leveraging user input as part of the correlation process.

Message correlation is challenging for several reasons. First, the definition of what constitutes a conversation is subjective. As an example, consider the interactions of a game service with its clients (players). One may be interested in grouping the messages based the session identifier into different sessions, in each players may play several games. Another may be interested to group the message based on the game identifier regardless of individual players involved. A third analyst may be interested in grouping the same messages based on player identifiers regardless of sessions and games that they have been involved in.

Even if the problem was clearly defined, we would still have to face the challenge of how to perform correlation. Simple approaches such as only looking at temporal vicinity cannot be applied, as we may have dozens of concurrent conversations at each point in time, and as the time gap between two invocations of a service in the context of a same conversation can range from seconds to days. Looking at message content is also far from trivial, as there is no obvious choice for identifying message attributes that can act as correlators, and as the correlation logic can be quite complex. For example, messages that are related to the same purchase order processing may all share the same order identifier, but it can also be that some messages include order identifiers, others include payment identifiers, but they are all related to the same order processing. In addition, not a single attribute but a set of attributes may be used to correlate messages to one another in a same conversation. Even if we knew that there is in the dataset a (set of) attribute(s) that can act as correlator, finding the appropriate one is not trivial, as there are many attributes that could be used as a basis for partitioning messages into conversations (e.g., consider the game service

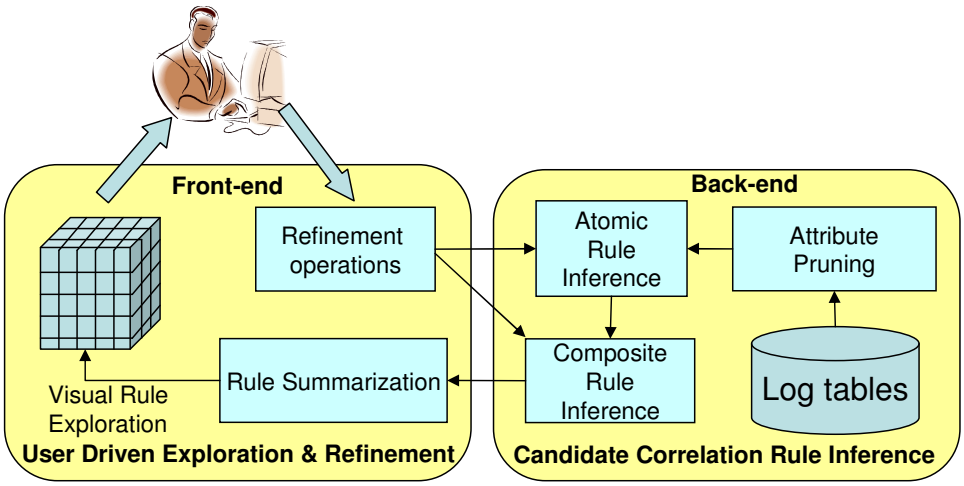

Figure 1: Overview of the correlation discovery process

data introduced above).

A final challenge consists in understanding how we can leverage possible user knowledge in performing the correlation and how we can guide users in the choice of a partitioning that is consistent with the user's correlation needs (which in turn depend on the kind of analysis to be performed on the correlated sets).

In general, performing message correlation therefore requires exploring different combination of a large number of message attributes and various possible correlation logics without a clear and formally identifiable goal.

In this paper, we propose a set of algorithms and techniques for message correlation. Figure 1 shows an overview of our proposed solution. It consists of two components: a back-end component that automatically discovers candidate correlators and a front-end component that provides a visual environment to explore the discovered correlations and refine them interactively. This paper, and the proposed solution, provides the following contributions:

1. We define and characterize the problem of message correlation, illustrating the different natures of the correlation problem and introducing the notion of *correlation dependency*, which identifies whether two messages are related, and why they can be considered as being related. These dependencies are expressed by means of rules, which are functions over message attributes.

2. We identify classes (patterns) of rules that can be used to define dependencies. For example, a simple rule is to identify correlation dependencies based on whether an attribute A has the same value in two messages, but more complex rules are possible. The identification of

these classes is based both on common ways of performing correlation as well as on recommendations by standardization proposals for how services should perform correlation.

3. We present a method for discovering correlation rules based on a level-wise approach [12]. We present a set of algorithms to analyze logs of messages and derive, for each rule class, a set of *candidate* rules that identify correlation dependencies among messages. We first exclude from the analysis those attributes that are irrelevant for correlation purposes, and then leverage the remaining attributes to derive the rules (steps *attribute pruning* and *atomic rule inference* in Figure 1). The algorithms operate based on (i) criteria and heuristics on the typical characteristics of conversations, and (ii) criteria and heuristics on typical characteristics of attributes whose values can be used as discriminators for defining membership of messages to conversations (and that therefore can be used to define correlation rules). As a simple example of such heuristics, we assume that a log will not contain as many conversations as there are messages, and will also not be composed of just one conversation comprising all messages. We expect that if a correlation rule exists, it will identify correlations to generate a proper partition of the log, where the number of conversations is more than one but less than the number of messages.

   The correlation dependency may be expressed using by a composition of rules rather than by a single atomic rule. For example, some messages within the same conversation are correlated via the orderID, others via a paymentID (step *composite rule inference* in Figure 1). We introduce an approach for atomic rule composition using conjunctive and disjunctive operators.

4. Due to the subjective nature of the correlation problem, the set of candidate rules is derived in a conservative manner, meaning that the algorithm tries to capture a wide range of correlation rules potentially of interest. To help the user in exploring the discovered rules, we define metrics computed based on the statistics of the conversations that are result of using a given rule to group messages in the log. We present a visual environment, in which rules are represented in a lattice. We also allow the user to refine the discovered correlation rules that leads to adding/removing rules to/from the set of discovered rules. In addition, we allow for the user knowledge to be incorporated into the correlation discovery approach as part of heuristics that drive the discovery.

In this paper, we also substantiate the arguments by means of experiments on real and synthetic datasets and we discuss our implementation. We believe that the proposed correlation discovery framework can have a

significant impact, also outside the area of services and conversations. For example, analogous concepts can be applied to correlation among process execution data (service messages are not significantly different from process execution events, and the problem of subjective interpretations of what constitutes a business process also applies), and hence can enable business process correlation and analysis throughout heterogeneous enterprise systems.

The reminder of the paper is structured as follows: Section 2 introduces some definitions and concepts and defines the correlation discovery problem. Section 3 presents our approach for automated discovery of correlation rules. In Section 4 we present our implementation and experimentations, conducted over three interaction logs. We discuss related work in Section 5. Finally, we conclude and discuss the future work in Section 6.

2 Message Correlation Problem

2.1 Web Service Logs

Messages are exchanged between services to fulfill a goal, e.g., to place a purchase order, receive an invoice, pay for good, and finally arrange for shipping. Taken together, these messages form a *conversation* that achieves a single business transaction. At any given time, there might be several ongoing conversations, corresponding to several clients interacting simultaneously with a service.

Messages exchanged during service conversations can be logged using various infrastructures [4]. For simplicity, we define in this article a generic log model where each message is logged as an event e, represented by a tuple of a relation $\mathcal{L} = \{e_1, e_2, \ldots, e_m\}$, and $e_i \in A_1 \times A_2 \times \cdots \times A_k$, where k is the number of attributes and m is the number of events. An event may have several attributes, corresponding to the attributes of the message exchanged. We denote by $e.A_i$, $1 \leq i \leq k$, the value of attribute A_i of event e. We further assume that each event e has at least three other attributes: the timestamp at which the event is recorded ($e.\tau$), the sender of the message ($e.s$), and the receiver ($e.r$). In the following, we use interchangeably *message*—the actual document exchanged between services—and *event*—their representation as tuples in the log.

Note that web service interactions usually involve structured (XML) messages (of different types, and hence with different attributes) organized in sections such as an header and one or more body parts. A preprocessing ETL-like is required to extract features from XML documents and represent them as event tuples (see Section 4). The attributes $A_1 \times \ldots \times A_k$ represent here the union of all the message attributes that belong to the different message types. Each message (event) will only have a subset of these attributes. We assume that there are some attributes that allow to determine if any two

message belong to the same conversation. We call such attributes *correlator attributes*, and the dependencies (relationships) defined over correlator attributes as *correlation rules*.

2.2 Correlation Rules

In this paper, we define a correlation of messages in a service log as a partition of log $\mathcal{L}$ into a collection of conversations $c_1, c_2, \ldots$, where each conversation c_i is a sequence of events (each corresponding to a message), denoted $c_i = \langle e_1, e_2, \ldots \rangle$. Note that, as we stressed in the introduction, correlation has a degree of subjectivity, so there is no "right" or "wrong" partitioning. The key toward a "good" partitioning lies in finding a way to group messages based on their relationship, for the sake of facilitating service execution analysis and protocol discovery.

The approach we follow is based on deriving a set of *correlation rules*, which are functions over pairs of correlator attributes for event pairs (e_x, e_y). Correlation rules specify whether events belong to the same conversation. A correlation rule R can alternatively be seen as a relation over $\mathcal{L} \times \mathcal{L}$ where $(e_x, e_y) \in R$ if and only if e_x and e_y are correlated. In general, a correlation rule can be defined as an arbitrary function over a pair of events. The precise form of a rule depends on the specific domain for which the rule is defined. We call *atomic* a rule defined on only one pair of attributes. For example, an atomic rule may specify an equality relationship between attributes A_i and A_i in (e_x, e_y), represented as $e_x.A_i = e_y.A_j$. A rule can be *composite* when it is made of a conjunction or disjunction of atomic rules.

The correlation discovery problem is the problem of inferring, from the data present in the log, how to correlate messages, that is, inferring the correlation rule definition. In this paper, we perform message correlation for the purpose of conversation discovery (reconstruction).

To derive correlation rules for messages, we first explore the different forms that correlation rules may take. In the context of message correlation, we leverage the different *patterns* in which dependency between messages manifests itself in service logs. This patterns are defined based on analysis of standard proposals and service implementations. Doing so, we restrict the rules to follow a relatively small set of corresponding *rule patterns*. Then, we analyze the log based on heuristics, that capture the subjective aspect of correlation to derive the candidate correlation rules and to determine which patterns can be applied.

2.3 Correlation Rule Patterns

In this section, we describe the correlation patterns commonly used for message correlation in web services.

Key-based Correlation. In many cases, conversations among services

are characterized by the fact that all messages have the same value for certain attributes. For example, when services adopt web services standards such as WS-Coordination, WS-Conversation or WS-CDL all messages may include a standard-specified attribute —typically called *(conversation or instance) identifier*— that acts as the correlation element. Even when these standards are not used, there may be an attribute that characterize the conversation. For example, messages related to a procurement conversation will likely have the same order ID, or the same pair $\langle supplierID, orderID \rangle$. Standard proposals such as WS-BPEL allow for specifying a given (set of) attribute(s) as correlator. In WS-BPEL the definition of correlation sets may be used for this purpose.

Based on this observation, we define the key-based correlation pattern, where some attributes of messages, called keys, are used to uniquely identify a conversation. As per keys in relational databases, correlation keys can be simple (single attribute) or composite (multiple attributes). For example, in the log presented in Figure 2(a), messages can be correlated using the simple key on attribute $ConvID$. In this case, the log contains two conversations $c_1 = \langle e_1, e_3, e_6 \rangle$ and $c_2 = \langle e_2, e_4, e_5 \rangle$. Figure 2(b) presents a log where correlation may be using a composite key on attributes $CustID$ and OID, resulting in conversations $c_1 = \langle e_1, e_3 \rangle$, $c_2 = \langle e_2, e_5 \rangle$ and $c_3 = \langle e_4, e_6 \rangle$. However, contrary to the traditional concept of key in database, the value of a key attribute is not unique *per tuple* in $\mathcal{L}$ but only it is unique *per conversation*. Nevertheless, we do not know which events in $\mathcal{L}$ form a same conversation a priori. As a consequence, it can not be known a priori if correlation has to be done using this composite key or with either of the simple keys $CustID$ and OID. All these three rules are valid candidate rules for correlating this log.

	msg	$ConvID$
e1	m1	1
e2	m1	2
e3	m2	1
e4	m2	2
e5	m3	2
e6	m3	1

(a)

	msg	$CustID$	OID
e1	m1	c1	o1
e2	m1	c2	o1
e3	m2	c1	o1
e4	m1	c2	o2
e5	m2	c2	o1
e6	m2	c2	o2

(b)

Figure 2: (a) `ConvID` as a simple key, (b) `CustID` and `OID` as composite keys

Rules corresponding to key-based correlation are of the form:

$$(e_x, e_y) \in R \Leftrightarrow e_x.A_i = e_y.A_i \wedge e_x.A_j = e_y.A_j \wedge \ldots$$

where $A_i, A_j, \ldots$ are the attributes composing the key.

Reference-based Correlation. Messages in a conversation may be related via links (references) that connect each message with a previous one in the same conversation. For example, a reply message is often correlated

with the request message (most reply messages in Web services contain a *reply to* attribute that links to the request). As another example, again taken from the procurement domain, all messages related to the purchase of a good will carry either the order ID, or the invoice ID, or both. In this case there is no common key (the invoice ID appears only in the later messages in the conversation), but there is always a way to link all messages in the conversation (except the first) to a previous one. For example, there is typically a message which includes both order ID and invoice ID, acting as bridge and closing the missing link in the link chain. This correlation pattern is supported by BPEL and can be defined using correlation sets. It also could be implemented using `RelatedTo` attribute in WS-Addressing standard that allows to refer to message ID (`MsgID`) of a previous message.

To account for this form of correlation, we introduce the reference-based pattern, in which messages of a conversation (except for the first one), are linked with a preceding one via a *reference attribute*. This pattern may have various sub-patterns. For instance, each message is linked to the preceding one in a conversation (*chain*). In Figure 3(a), events form the sequence $\langle e_1, e_3, e_5 \rangle$ where events of the pair (e_1, e_3) are linked by a common value on attribute OID (Order Id) and the pair (e_3, e_5) by a common value on attribute IID (Invoice Id). Another sub-pattern is the one where all messages are linked to the first message. For instance, in Figure 3(b), events e_3 and e_5 can both be correlated with event e_1 since their value on attribute Ref is equal to that of e_1 on attribute OID.

Note that the attribute used for reference-based correlation may change during a conversation. Furthermore, even the value used for reference correlation may be common only to pairs of messages, not to the entire conversation. Hence, key-based correlation is not a particular case of reference-based correlation. Similarly to key-based correlation, references may be defined over more than one attribute.

If the key attribute(s) is known, then reconstructing the conversations in a key-based correlation consists in a standard `GROUP BY` operation. In reference-based correlation, this is not the case and a transitive closure of the relation between attributes has to be computed.

	msg	OID	IID	PID
e1	m1	o1		
e2	m1	o2		
e3	m2	o1	i1	
e4	m2	o2	i2	
e5	m3		i1	p1
e6	m3		i2	p2

(a) chain

	msg	OID	Ref
e1	m1	o1	
e2	m1	o2	
e3	m2		o1
e4	m2		o2
e5	m2		o1
e6	m2		o2

(b) tree

Figure 3: Reference-based correlation

For the chain example of Figure 3(a), a possible correlation rule would be $R_{chain} : e_x.OID = e_y.OID \vee e_x.IID = e_y.IID$. It would be $R_{tree} : e_x.OID = e_y.Ref$ for the tree example of Figure 3(b).

Time constraints on correlation. In some cases, it is not sufficient to specify a correlation rule using equalities of attribute values to distinguish between conversations. For example, in the case of a key-based correlation, key values may be reused (recycled) over time and, thus, a key value uniquely identifies a conversation only within a certain time window. Intuitively, messages that are close in time have higher probability of being logically related (be dependent). Hence, the correlation logic may have to include temporal vicinity between events (though this is rarely a rule that can be used by itself, since time can aid the definition of correlation but can rarely be used as the only correlator, especially when it is possible to have many concurrent conversations between a service and a same client).

Correlation rules would have to account for this situation with an additional condition on the time difference between two messages, e.g. $R : |e_x.\tau - e_y.\tau| \leq \theta$, where θ represents a maximal time separation between two messages of a same conversation. It is also possible to specify the maximum duration of a conversation (the different of time between the first and the last message of a same conversation). WS-CDL standard proposal supports definition of time constraints on correlation rules. We do not discuss this issue further in this paper.

Combination of the above patterns. In general, correlation rules may include a combination of atomic rules, of the same or of different patterns. For instance, messages may first be correlated in a key-based way while customer query for product catalogs or product description. Then, after an order is placed, subsequent messages are correlated with reference to that order. We discuss composition of rule patterns in Section 3.3.

3 Automated Discovery of Correlation Rules

The automated correlation rule discovery consists of three steps: (i) candidate attribute selection, which introduces techniques to select attributes of $\mathcal{L}$; (ii) atomic rule discovery, in which candidate attributes from the previous step are used to identify atomic candidate rules; (iii) composite rule discovery, in which techniques for composing atomic rules are presented. We adopt a level-wise approach [12] for generating, pruning and exploring the space of potentially interesting correlation rules.

3.1 Candidate Attribute Selection

All attributes of log $\mathcal{L}$ are initially considered as candidates. Then, we apply a number of heuristics to prune obvious non-candidates. Here, we first discuss the heuristics and then explain the attribute selection techniques.

3.1.1 Heuristics

From the correlation patterns identified in Section 2.3, it can be seen that attributes used in key-based correlation or reference-based correlation are both some sort of "identifier" (code) attributes. While there can be a variety of identifier domains (e.g. string for login names, or integer for customer codes), these attributes share some common aspects, in particular with respect to identifiers that are potentially interesting for correlation purposes:

1. The domain is nominal (string or number). When a number is used, it is typically an integer. This allows to exclude for example attributes containing floating point values. Similarly, free text attributes can be excluded. In our approach, we differentiate the free text from string identifier on the basis of the length of the string, and also the number of words by setting a threshold (see Section 4).

2. The number of distinct values is rather large, as it needs to discriminate among many different entities, e.g., orders, invoices, or payments. When the number of values is small (e.g., identifiers of the banks that a company uses for payments), these identifiers are often used in conjunction with other (more discriminating) identifiers to form, for instance, a composite key. Hence, it is reasonable to avoid considering attributes whose domain is too small when compared to the dataset size. Furthermore, in general, the number of distinct values is not fixed and depends on the log size (the higher the number of conversations, the higher the number of distinct order IDs).

3. Values are repeated in the dataset. To be useful for correlation purposes, a same value has to appear in at least in two messages (reference-based correlation). Values that are unique in the dataset may reflect partial conversations (i.e. only the first message has been logged) but if a majority of the values of an attribute appear only once (in this attribute or others) then it is more likely that this attribute is not an identifier useful for correlation.

3.1.2 Correlator Attribute Selection

We now show how the above heuristics are put at work in identifying candidate correlator attributes in a key or reference based correlation rule.

First, we eliminate any attribute whose type is either floating point or that contains long strings. We have set the threshold for what constitutes a long string arbitrarily to 100 characters. Note that other methods could be used to more precisely differentiate between free text and potential correlators such as, for instance, analyzing the content of the text. Alternatively, a manual selection of the attributes could prove very effective and would not be too time consuming.

Then, we count the number of distinct values of each attribute A_i relative to the total number of values ($distinct\_ratio(A_i) = d/n$, where d is the number of distinct values, and n denotes the total number of non-null values for this attribute). $distinct\_ratio(A_i)$ defines a measure to decide if an attribute is a non-candidate. In fact, based on the heuristics above, there are two types of attributes that are not interesting:

- Attributes with categorical domains: Their characteristic, compared with free text or identifiers, is to have only few distinct values. We identify them by setting an arbitrary threshold called $lowerThreshold$ that defines a minimum for $distinct\_ratio$.

- Attributes that do not have repeated values: according to heuristics, values of candidate correlator attributes have to be repeated somewhere in the dataset, either in the same attribute or in another one. We can expect that most distinct values will appear at least twice since they correlate at least two messages. In the case of value repeated in a same attribute, $distinct\_ratio$ will be well below one. On the other hand, if values repeat in other attributes, then we should observe a large number of tuples with a null (empty) value on that attribute. Indeed, if all value are non-null and they do not repeat, this attribute would lead to correlate messages in conversations of a single message. Hence, we can set an upper threshold for $distinct\_ratio$ (called $upperThreshold$ hereafter) assorted with a condition on the ratio of null values to the number of non-null values.

Thresholds are difficult to set. However, we only need to eliminate attributes that are *obviously* non-candidate. Therefore, we can adopt a pessimistic approach and set these thresholds to the lowest (resp. highest) extreme values so to reject only attributes for which the classification is clear (See section 4 for how these values are set). Such way of settings does not create problems as we accept more irrelevant attributes, however, it does not impact significantly the performance of the process nor its final result.

3.2 Atomic Rule Discovery

In the previous subsection, we identified the candidate correlator attributes. In this section, we present an approach for discovering atomic candidate correlation rules. The atomic rule discovery process is based on finding the intersection of distinct values of attribute pairs in $\mathcal{L}$. We then present an approach based on some general heuristics to prune non-candidate atomic rules.

3.2.1 Candidate Discovery Approach

As the focus of this paper is on discovering key-based and reference-based rule patterns for conversation reconstruction, as the first step, we look for atomic candidate rules of form $R : e_x.A_i = e_y.A_j$, $1 \leq i \leq j \leq k$ over event pairs $(e_x, e_y) \in \mathcal{L}^2$. Such rules define equality relationships between pairs of attributes for each event pair (e_x, e_y). To identify whether a given candidate rule holds, we need to compute the support of each rule. The support shows the number of pairs (e_x, e_y) for which the values of attributes $e_x.A_i$ and $e_y.A_j$ are equal. A high support implies that the correlation rule holds for a large subset of the dataset and that equal values found in these two attributes is not a mere coincidence.

If the support of a given rule R is non zero, then it means that there are values that appear in both attributes A_i and A_j in $\mathcal{L}$, i.e., the intersection of values of A_i and A_j is not empty. So, instead of actually computing the support of rules, which would be computationally expensive, we compare the sets $distinct(A_i)$ and $distinct(A_J)$ of distinct values of A_i and A_j respectively. Moreover, $distinct(A_i)$ was already computed during the candidate attribute identification phase (see previous section) for all attributes and so, in our implementation, these sets are computed only once. An initial list of candidate rules is produced from all the pairs $(A_i, A_j), 1 \leq i \leq j \leq k$ such that $|distinct(A_i) \cap distinct(A_j)| > 0$ holds. For example, for the table in Figure 3(b), the discovered rules and their support are: $OID = OID : 2$, $IID = Ref : 2$.

3.2.2 Heuristic-based Atomic Rule Pruning

We use the following heuristics to decide if a rule is a non-candidate: If attributes A_i and A_j define a reference-based correlation $(R : e_x.A_i = e_y.A_j)$, then it is likely that, typically, values in A_j also appear in A_i, and vice versa. If by contrast A_i and A_j have only few distinct values in common and many that are different, then we can conclude that this pair is not correlated, and the relationship among this two attributes is only incidental. So, it does not form a valid rule and it can be removed from the list of candidate correlation rules.

In order to reflect this idea, we define a measure of *attribute-pair interestingness* denoted by $\Psi(A_i, A_j)$:

$$\Psi(A_i, A_j) = \frac{|distinct(A_i) \cap distinct(A_j)|}{max(|distinct(A_i)|, |distinct(A_j)|)}$$

$\Psi(A_i, A_j)$ is defined over $[0, 1]$ with 1 (resp. 0) indicating a pair of attributes that share a large number (resp. only few) of their values. We set a threshold ψ and prune all rules for which $\Psi(A_i, A_j) < \psi$. Note that this measure is used only for pairs (A_i, A_j) with $i \neq j$. Indeed, attributes

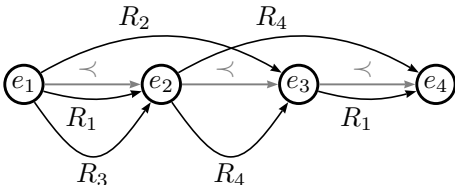

Figure 4: Rules graph of candidate rules

not pruned in the attribute pruning step (Section 3.1) are already shown as candidate for a key-based correlation.

3.3 Composite Rule Discovery

Correlation rules may not always be atomic, but composed of several atomic rules to form composite rules (see Section 2.3). In this paper, we examine two types of composition operators: *conjunctive* ($\wedge$), used for representing keys defined on more than one attributes, and *disjunctive* ($\vee$), used to identify conversations where messages are not all correlated using the same rule.

In order to discover all the candidate composite rules, we take as input the candidate atomic rules inferred in the previous section and proceed in the following steps:

1. We build a list of candidate conjunctive rules. This is done by testing the possible combinations of atomic rules and pruning obvious non-candidate combinations. After this step, conjunctive rules are treated as atomic rules.

2. The candidate atomic are combined in disjunctions and organized in a lattice structure.

Before presenting these two steps, we discuss how rules are used to partition a log and define a series of metrics used to characterize a log partition.

3.3.1 Partitioning the Log

In order to illustrate how correlation rules (atomic or composite) partition the log into conversations, we will represent the relationships among events as a graph $G_R = (\mathcal{L}, R)$. In this graph, nodes represent events and there is an edge from event e_x to event e_y if and only if $(e_x, e_y) \in R$. Building edges in this way for candidate atomic rule in G_R, we obtain a labeled multigraph called *rule graph* denoted by G. For example, assume that we have a log $\mathcal{L}$ containing four events $\mathcal{L} = \{e_1, e_2, e_3, e_4\}$, and that we have discovered four atomic rules R_1, R_2, R_3, R_4, a possible rule graph for this log is illustrated in Figure 4.

Partitioning a log $\mathcal{L}$ with a correlation rule R consists of identifying the set of conversations $\mathcal{C}_R(L) = \{c_1, c_2, \ldots\}, c_i \subseteq \mathcal{L}$ such that

$$\begin{cases} \forall c_i \in \mathcal{C}_R(L), \quad e_x \in c_i \Leftrightarrow \exists e_y, (e_x, e_y) \in R \vee (e_y, e_x) \in R \\ \forall c_i, c_j \in \mathcal{C}2_R(L), \quad i \neq j \Leftrightarrow c_i \cap c_j = \emptyset \end{cases}$$

Thus, identifying the conversations $\mathcal{C}_R$ can be formulated as identifying the connected components of G_R. For example, correlating the rule graph presented in Figure 4 using rule R_1 produces two connected components, each of length 2 and we have

$$\mathcal{C}_{R_1} = \{\langle e_1, e_2 \rangle, \quad \langle e_3, e_4 \rangle\}.$$

The problem of identifying the connected components of a graph is well-investigated in the literature [2]. In the database context, this problem consists in identifying the transitive closure of a query. Since we can not be sure that the components do not have cycles, we chose to implement this decomposition using a breadth-first search approach. In order to optimize this processing time as well as the count of distinct values of attributes that is used in Section 3.1 and 3.2, we first index separately each attribute of the log. With this setting, our experiments have shown that partitioning the log is very fast (see Section 4).

A number of metrics can be defined to characterize a partition $\mathcal{C}_R$ in a concise way:

- $AvgLen(\mathcal{C}_R)$, $MinLen(\mathcal{C}_R)$ and $MaxLen(\mathcal{C}_R(\mathcal{L}))$ represent respectively the average, minimum and maximum length of conversations in conversation $\mathcal{C}_R$;
- $ConvCount(\mathcal{C}_R)$ represents the number of conversations. It is equal to $|\mathcal{C}_R|$;
- $UncorrelatedCount(\mathcal{C}_R)$ represents the number of messages that are not related to any other message according to R.

In the following, we explain how these metrics are used to reduce the search space for candidate composite rules.

3.3.2 Conjunctive Rules

Composite keys are defined by equality of values on more than one attributes. For example, if purchase order numbers are uniquely assigned on a per customer basis, it is not sufficient to consider only the purchase order number to correctly correlate messages. It is necessary to use both the purchase order number *and* the customer number to relate a message to the correct conversation (see Figure 2(b)). The operator $\wedge$ defined hereafter will be used to express this type of situation.

Consider two correlation rules R_1 and R_2. The composite rule $R_{1\wedge 2} = R_1 \wedge R_2$ is defined as follows:

$$\begin{aligned}(e_x, e_y) \in R_{1\wedge 2} &\Leftrightarrow (e_x, e_y) \in R_1 \wedge (e_x, e_y) \in R_2 \\ &\Leftrightarrow (e_x, e_y) \in R_1 \cap R_2\end{aligned}$$

in this expression, R_1 and R_2 are both atomic rules and they each express that events e_x and e_y are equal on some attributes.

We need to identify all the candidate conjunctive rules. The number of possible conjunctions from r atomic rules is large, i.e., $2^r - 1$. In the following we propose three criteria that allow to avoid computing a large portion of them:

Attribute definition constraints: Taken individually, R_1 and R_2 could represent a key or reference based correlation. Thus, each of the attributes used in R_1 and R_2 may be undefined for some events (see Figure 3). However, when considered together in a conjunction, a new constraint appears: attributes of R_2 have to be defined whenever the attributes of R_1 are defined. More formally, a general expression of $R_{1\wedge 2}$ is of the form

$$(e_x, e_y) \in R_{1\wedge 2} \Leftrightarrow e_x.A_{i_1} = e_y.A_{j_1} \wedge e_x.A_{i_2} = e_y.A_{j_2}.$$

The above constraint implies that for any message of the log, attribute A_{i1} is defined if and only if A_{i_2} is also defined and that A_{j_1} is defined if and only if A_{j_2} is also defined. Thus, conjunctions are only valid for rule pairs that satisfy this constraint, i.e., they are defined only on the same set of messages.

Identifiers have repeating values: A conjunctive rule results in a new identifier (for a reference or a key) built using the attributes of each atomic rule. Using heuristics similar to those in section 3.1, we know that identifiers have to appear in more than one message to be valid. For example, the correlation rule $R_{1\wedge 2}$ defined above is valid only if most of the pairs of values that are defined on attributes (A_{i_1}, A_{i_2}) also appear in at least one other message on attribute (A_{j_1}, A_{j_2}) in any order.

Inclusion Property. In graph G, if the set of edges of rule R_1 is included in those of rule R_2, i.e., $R_1 \subset R_2$:

$$\forall (e_x, e_y) \in R_1 \Rightarrow (e_x, e_y) \in R_2$$

then, we have $R_1 \wedge R_2 = R_1$. Hence, it is not needed to compute $R_1 \wedge R_2$. Furthermore, if $R_1 = R_2$, i.e.,

$$\forall (e_x, e_y) \in R_1 \Leftrightarrow (e_x, e_y) \in R_2$$

then, R_1 and R_2 partition the log in the same way, so it is enough to compute the log partitioning for one of them and then to extend the result

to the other. This implies that one further rule is punned from the set of rules that its conjunction with other rules to be computed.

Monotonic property of partitioning metrics: An important property of metrics of a partition is the *monotonicity* with respect to the conjunctive operator. A conjunctive operator is indeed an intersection of the relations defined by the two rules. Hence, applying a conjunction will either leave unchanged or reduce the number of neighbors of a message, i.e., the number of messages to which it is related. This makes conversations build on conjunctive rules both shorter and more numerous than conversations built on the atomic rules and we have

$$\begin{cases} MinLen(\mathcal{C}_{R_{1\wedge 2}}) \leq \min(MinLen(\mathcal{C}_{R_1}), MinLen(\mathcal{C}_{R_2})) \\ ConvCount(\mathcal{C}_{R_{1\wedge 2}}(\mathcal{L})) \geq \max(ConvCount(\mathcal{C}_{R_1}(\mathcal{L})), \\ \qquad\qquad\qquad\qquad ConvCount(\mathcal{C}_{R_2}(\mathcal{L}))) \end{cases}$$

With the exception of conversations that are partially logged, most of the conversations should have a length of two or more. In other words, the number of conversations should be less than half of the number of messages. Thus, if a pair of rules does not satisfy the above criteria, it is unnecessary to compute their conjunction. Since *ConvCount* is monotonically non-increasing with conjunctive, it is not necessary to examine conjunctions of bigger size (with more rules) when these rules themselves do not satisfy this criteria.

The criteria are applied after the first three criteria since it requires to actually partition the log, which is more computationally expensive than testing for existence of values in an attribute (a selection operation) as in the first two criteria.

The set of candidate conjunctive rules is the set of possible conjunctive rules for which all the above three criteria hold. Note that for all these criteria, if the constraints do not hold for a conjunction of two rules, then it will not hold for conjunctions of a larger number of rules.

3.3.3 Disjunctive Rules

Disjunction of rules are useful for expressing situations where messages are not correlated always the same way throughout a conversation. This is the case, for example, when an invoice message references a purchase order number while a payment message references the invoice number (see Figure 3(a)). To correlate these three messages, we need to define two rules.

Given two atomic or conjunctive rules R_1 and R_2, the disjunctive rule $R_{1\vee 2}$ is defined as follows:

$$\begin{aligned} (e_x, e_y) \in R_{1\vee 2} &\Leftrightarrow (e_x, e_y) \in R_1 \vee (e_x, e_y) \in R_2 \\ &\Leftrightarrow (e_x, e_y) \in R_1 \cup R_2 \end{aligned}$$

Since disjunctions are built based on both atomic and conjunctive rules, it is not possible to apply any attribute level criterion to reduce the search space.

Inclusion Property: If relation R_1 is included in R_2, i.e., $R_1 \subset R_2$, then, we have $R_1 \vee R_2 = R_2$. Hence, it is not needed to compute $R_1 \wedge R_2$. Furthermore, when $R_1 = R_2$, as stated before, R_1 and R_2 partition the log in the same way and $R_1 \wedge R_2 = R_1 = R_2$. Hence, so it is enough to compute the log partitioning for one of them and then to extend the result to the other.

Monotonic property of partitioning metrics: The partitioning metrics are also monotonic with respect to the disjunctive operator: the more we have rules in a disjunction, the more each message becomes connected with others and the longest are the conversations in the corresponding partition. We have:

$MaxLen(\mathcal{C}_{R_{1\wedge 2}}) \geq \max(MaxLen(\mathcal{C}_{R_1}), MaxLen(\mathcal{C}_{R_2}))ConvCount(\mathcal{C}_{R_{1\wedge 2}}) \geq \max(ConvCount(\mathcal{C}_{R_1}),$

A disjunction of rules is said to be valid if it leads to a partitioning of the log in at least two conversations. Since $ConvCount$ is monotonically non-decreasing with disjunction, it is not necessary to examine disjunction comprising more rules when these rules themselves do not satisfy this criteria.

In theory, if there are r candidate atomic rules, then there are $cr = 2^{r-1}$ possible conjunctive rules. Let dr denote the number of possible disjunctive rules, we have $dr = 2^{cr+r-1}$, as conjunctive rules are considered atomic. However, as we will see in the experiments (Section 4), applying the introduced criteria and heuristics shrinks this search space significantly.

3.3.4 Rule Lattice

When composing together in disjunctive form both atomic and conjunctive rules, we can organize rules in a lattice structure to facilitate user navigation. An example of rule lattice is illustrated in Figure 5.

The bottom layer of nodes corresponds to both the atomic and conjunctive rules identified in Sections 3.2 and 3.3.2. Each higher level layer represents a combination of the lower level rules. Nodes that are shown in dark gray are nodes that are not computed. They correspond to two types of rules:

- Rules combining one or more lower level composite rules found invalid. Suppose that rule $R_2 \vee R_3$ (highlighted in Figure 5) is computed and that the corresponding partitioning $\mathcal{C}_{R_{2\vee 3}}$ produces a single conversation comprising all the messages of the log. Extending this disjunction to additional rules will always result in the same conversation due to monotonicity property (see Section 3.3.3) and thus it is unnecessary to compute it.

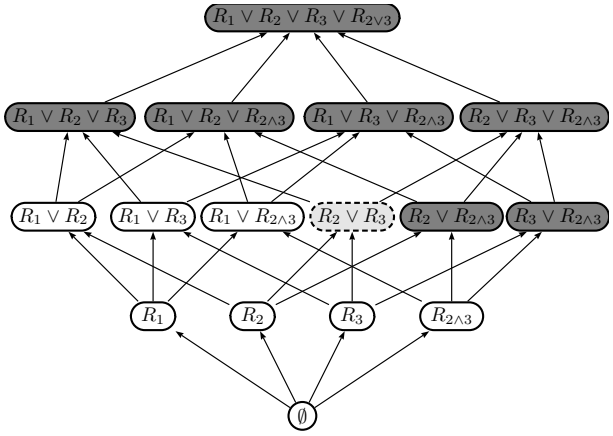

Figure 5: Candidate Correlation Rule Lattice

- Rules combining disjunction and conjunction of a same atomic rule. For example, the correlation rule $R_2 \vee R_{2\wedge 3}$ is equivalent to the correlation rule R_2. They are not useful since conjunctive and disjunctive operators are here associative (they correspond to intersection and union of relations).

With this simple example, it can be seen that as soon as a rule is identified as invalid, this propagates to the upper levels of the lattice. In the next section, we present our experiments and in particular, we discuss the number of composite rules that were actually computed. This number is very low when compared to the number of possible combinations.

4 Implementation and Experiments

We have implemented the correlation discovery approach presented in this paper in a prototype tool as part of a larger project, called ServiceMosaic, a framework for analysis and management of Web service interactions. It has been implemented using Java 5.0 as Plug-ins in Eclipse framework, and with PostgreSQL 8.2 as the database engine. All the experiments have been performed on a notebook machine with 2.3 GHz Duo Core CPU, and 2GB of memory.

4.1 Datasets

We have conducted the experiments on both synthetic and real-world datasets, described next.

Dataset	*Retailer*	*Robostrike*	*PurchaseNode*
# of messages	10	32	26
# of events	2,800	4,776,330	50,806
# of attributes	15	98	26

Table 1: Characteristics of the datasets

4.1.1 Synthetic Dataset

We used the interaction log of a *Retailer* service, developed based on the scenario of WS-I[1] (the Web Service Interoperability organization). In this dataset, the log is collected using a real-world commercial logging system for Web services (HP SOA Manager[2]). The Retailer service is implemented in Java and uses Apache Axis as SOAP implementation engine and Apache Tomcat as Web application server.

Table 1 shows the characteristics of this dataset. The log has 2800 tuples, each corresponding to an operation invocation. These messages form 480 conversations and are correlated using a key-based approach. HP SOA Manager records information organized in 13 attributes. Examples of attributes are $SequenceID$, $ServiceName$, $Operation$, $Timestamp$, $RequestSize$, and $ResponseSize$. In addition, we also extracted two attributes from the body of XML messages, namely $RequestID$ and $ResponseID$. In fact, both $RequestID$ and $ResponseID$ contain the same value and each value is a conversation identifier.

4.1.2 Real-world Datasets

We used two real-world datasets: the interaction log of a multi-player online game service called *Robostrike*[3], and a workflow management system log corresponding to a purchase order management system, called *PurchaseNode*.

Robostrike service exposes 32 operations invoked by sending/receiving XML messages. Table 1 presents the statistics of this dataset. The log contains more than $4,700,000$ events. In a pre-processing stage, we extracted all the attributes of all messages. To this end, the XML schema was flattened and each element was given a distinct element name.

The PurchaseNode dataset was originally organized into two tables: one for the workflow definitions and the other for the workflow instances (execution data). The workflow definition table defines 14 workflows using different combinations of the 26 workflow tasks. For this experiment we joined the two tables, based on workflow identifier, to produce a log of $50,806$ records.

[1] www.ws-i.org

[2] managementsoftware.hp.com/products/soa/

[3] www.robostrike.com

Dataset	*Retailer*	*Robostrike*	*PurchaseNode*
initial # of attribute	15	98	26
# attributes after pruning	2	13	3
# of atomic rules	3	31	6
# of atomic rules after pruning	3	8	4
# of discovered composite rules	4	57	3

Table 2: The performance of the automated correlation discovery process

By using this dataset we also test the applicability of the approach to workflow data.

4.2 Automated Correlation Rule Discovery

We ran experiments to validate the performance of the proposed approach on large datasets. We separately report the results of each of the three steps of the correlation discovery process.

4.2.1 Candidate Attribute Selection

In this step, we compute the *distinct_ratio* defined in Section 3.1.2. On the basis of a *LowerThreshold* (resp. *UpperThreshold*) of *distinct_ratio*, we select attributes that have neither too much nor too many) distinct values (see Section 3.1.2). Our purpose in this experiment is twofold: (i) validate if this heuristics are effective to identify non-correlator attributes even in the case of a pessimistic setting of thresholds, and (ii) verify the time needed to compute this ratio for all attributes.

Table 2 shows the summary of the result of applying this attribute selection approach to the datasets. In this settings, we used $upperThreshold = 0.9$ and $lowerThreshold$ such that the number of distinct values is at least 15 (i.e. we assume the dataset contains at least 15 conversations). This setting is designed to prune only attributes which are *obvious* non-candidate since other steps of the process can further, and more precisely (i.e. without requiring threshold settings) identify attributes which are not correlators.

For *PurchaseNode*, out of 26 attributes, 21 attributes are pruned on $lowerThreshold$, and one attribute based on $upperThreshold$. Only 3 attributes remained. For the *Robostrike* dataset, out of 98 attributes there were 13 attributes that were selected. Finally, from the *Retailer* dataset only 2 attributes out of 15 were selected, namely $RequestID$ and $ResponseID$.

Computation is performed with simple requests. For one million records taken from the *Robostrike* dataset, computing *distinct_ratio* for all the 98 attributes took 19 minutes. Note that we perform this evaluation on a table where each attribute is indexed.

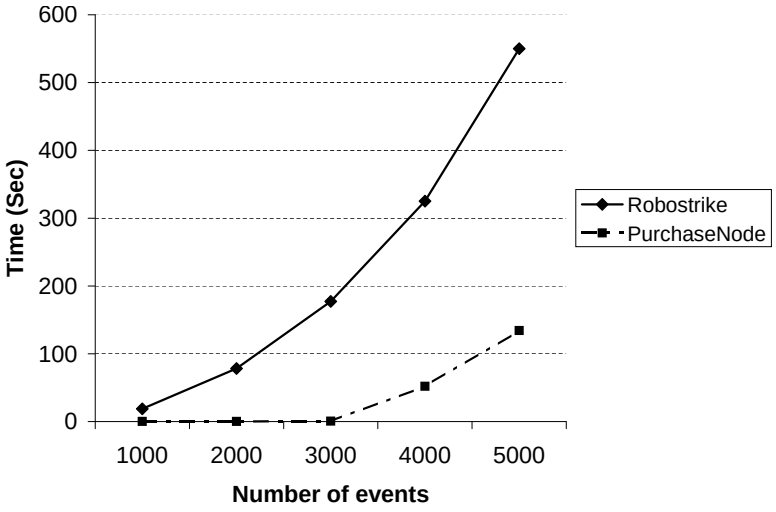

Figure 6: The execution of time of composite rule discovery

4.2.2 Atomic Rule Inference

The approach presented in Section 3.2.2 for rule inference is based on identifying the number of shared values between distinct values of different pairs of attributes. The execution time for computing the intersection of attributes is rather small. For example, for the *Robostrike* dataset and for 1,000,000 records, the execution of this step takes slightly more than 400 seconds. The experiments show that this time is linearly increasing with the growth in the dataset size.

For the *PurchaseNode* dataset, out of 6 candidate atomic rules, 4 remain after pruning. In case of the *Robostrike* dataset, 31 rules were discovered, out of which 8 identified as candidate atomic correlation rules. Finally, for the *Retailer* dataset, all three candidate atomic rules qualified. These results are summarized in Table 2. It should be noted that for the computation of atomic rules in all these experiments, the entire logs were used (e.g., 4 millions events for the *Robostrike* dataset).

4.2.3 Discovery of Composite Rules

Figure 6 shows the execution time of the algorithm for computing the composite rules based on atomic rules in the three datasets, with various dataset size. It takes just under 10 minutes to compute the composite rules for 5000 events in the *Robostrike* dataset. The execution time of the algorithm for the *Retailer* dataset for 2800 events present in this table takes 5 seconds.

It is interesting to note that the dataset size does not notably impact the number of composite rules discovered. For the *Robostrike* dataset, from

the 8 atomic rules discovered in the previous step, the theorical number of combination of rules was 2^8 (considering only disjunctions). After completion of the discovery process, this number was only 57. Then, we performed a similar computation with $500,000$ and then with $1,000,000$ events on this same *Robostrike* dataset. This tenfold increase in size does not impact significantly the number of correlation rules discovered: the results were respectively 58 and 59, among which 57 are the same as the 57 rules discovered with 5000 events.

For the *Retailer* dataset, which follows a key-based correlation, all atomic rules discovered are keys. The key to choose depends on the application of the correlation. For $PurchaseNode$, the rule discovery process correctly identified $FlowInstanceID$ as a key attribute. This key however is only one of the 7 possible correlations.

The results of the correlation discovery are more interesting on the *Robostrike* dataset. The various rules discovered correspond to various levels of analysis of the game. For instance, it is possible to correlate as conversation a single game, or a long session that comprises several games. Another correlation is possible at the user level, regardless of his session. Finally, the algorithm identified a surprising correlation based on chat conversations among groups of players. These correlation mixes key-based and reference-based correlation. For instance, messages during a game refer to the game number while messages regarding a single player are of key type, based on her user id.

In the next section, we discuss how the computation of metrics for each rule allows to explore the set of discovered rules to look for the desired correlation.

4.3 User Driven Exploration and Refinement of Correlation Rules

In this section, we present the front-end component of the system in two parts: correlation rule summarization, and refinement interface (see Figure 1).

4.3.1 Correlation Rule Summary

The discovered rules can be structured in a lattice, as discussed in Section 3.3.4, which facilitates navigation among candidate rules. To help users identify interesting correlation rules, each node is displayed with a summary that describes the properties of its corresponding correlation rule. The summary comprises the following information:

- *Metrics*: These correspond to the aggregate values defined in Section 3.3.1. We also display an histograms that shows the distribution of conversation lengths.

- *Description*: The rule definition is displayed as in the notation similar to that used in this paper (the difference is that we only display the attribute names). When attribute names carry some semantic, these rules are interesting in themselves and help understand the nature of conversation discovered. These in particular proved useful in the case of the game dataset to differentiate among the 57 candidate rules.

4.3.2 Refinement Interface

We propose two refinement operations that allow users to further expand or reduce the search space. Following refinement operations are available:

- Users can propose new rules to be examined. This operation is handled by adding a new atomic rule to the set of existing atomic rules. Adding this atomic rule automatically updates the lattice to generate additional nodes corresponding to composition of this rule with other atomic rules.
- Users can propose to remove a rule from the lattice. This operation updates the lattice to remove composite rules built by composition of this rule.

4.3.3 User Domain Knowledge

We observe that the user may have some domain knowledge to help in reducing the space of possible correlations, i.e., to do not computing some nodes in the lattice or to identify interesting rules for the user. There are various types of user knowledge information, among which the most general ones include:

- The average number of conversation per day. For example, the user may be able to tell how many purchase order are lodged per day.
- The average/minimum/maximum number of messages per each conversation, if possible;
- an estimate of the number of conversation in the given log; This and above two items help in deciding on interestingness of a discovered rules.
- Attributes that potentially are (not) used for correlation; This is helpful for attribute pruning step.
- the rule pattern that is used for correlation of the log.

- The user may have a log with millions of records. However, parsing all events in the system after a certain number of events may not lead to discovering further rules. For the sake of processing time, users can suggest to use part of log, for instance, all messages between a certain period of time, e.g., 3 months, if it is assumed that there are enough conversations and also the longest possible conversation will not take more than 3 months.

Our framework allows to incorporate above domain knowledge as part of heuristics used during discovery. In particular, if the user knows the correlation rule pattern that is used for correlating the messages in the log, it has the following implications on the discovery approach:

Key-based Pattern. This knowledge helps in two ways. First, we can define an additional heuristics that helps in pruning non-candidate attributes: the property of a key-based attribute is that is has some non-null value over all messages in the log. So, among candidate attribute with proper set of distinct values we only consider the ones that the number of their values is equal to the number of messages in the log. This reduce the candidate space significantly.

As discussed in Section 2.3, the key is either simple or composite. Discovered atomic rules are candidate for simple keys. As the second implication of this knowledge, for discovering composite key attributes, we only need to apply the conjunctive operator ($\wedge$) on atomic rules. This means that the search space for composite rules is significantly reduced. If there are r atomic candidate rules, then the number of possible composite (conjunctive) rules is $cr = 2^{r-1}$. Note that $cr \ll dr$ (see Section 3.3.3).

Reference-based Pattern. Knowing the the log is correlated using a reference-based pattern helps in further pruning non-candidate attributes based on the heuristics that for key-based attributes the number of values is equal to the number of messages in the log (having non-null values in all messages). This means that we also prune the attributes that are candidate for key-based correlation. However, the method for computing composite keys is the same, as it is possible to use both conjunctive and disjunctive rules. Conjunctive rules allow for discovering composite reference attributes, and disjunctive are required to discover composite correlation rules to allow each pair of messages in a conversation to be correlated using a different rule.

5 Related Work

There is a large body of related work in the database area. In this section, we discuss how our work differentiates from other research.

Functional dependencies and composite keys. The problem of correlation dependency analysis is related to that of discovering functional

dependency. In functional dependency inference [10, 16] the problem is to identify properties of the form $A \rightarrow B$, where A and B are sets of attributes of the relation, that hold for all or a well defined subset of tuples of this relation. Discovering a functional dependency is interesting as it reveals an implicit constraint over attribute values of tuples, which are not expressed in the schema. The main difference of this problem and correlation discovery is that correlation is not expressed *over* the tuples of a database but *among* the tuples and at the conversation level, which is not known a priori. Messages of a log may include valid functional dependencies, but these dependencies do not reveal anything about the conversations. Indeed, each group of messages that form a conversation exhibit a very strong dependency, however, this dependency is formed after conversations are known, i.e., the correlation rule is discovered.

The problem of discovering (composite) keys in relational databases [17] is related to that of message correlation: both seek to identify a collection of attributes that identify individual instances. However, two major differences make the solution to the former unsuitable for the latter. First, the validity of candidate composite keys can be assessed objectively since, by definition, each key value shall identify a unique instance. By contrast, there are only heuristics—but no definite criteria—to decide on the validity of correlation of messages. Second, keys are only one of the various *rule patterns* (see Section 2.3) that may be used for the correlation of messages.

Association Rule Mining. Association rule mining techniques identify values that co-occur frequently in a data set [8]. Such rules identify, for example, the likelihood of a customer that bought product A, to also purchase product B, or what other set of products a customer is likely to buy, given that she bought product D. Although such rules have links to the notion of correlation and implication (causality), some correlation patterns can not be identified using this approaches. Consider the case of reference-based correlation where a pair of attributes is used to link messages in a chain. For this two attributes, which can be defined for all tuples, values never appear concurrently in the same event. Moreover, on the overall database, each value occurs only twice, one in each message of the chain, and hence is not frequent. Another intuition is that for item set to be frequent, the collection of possible values have to be defined on a finite, and relatively small, domain. In the case of correlation rules, we are looking for attribute that express identifier, i.e. values that repeat at most in message of a single conversation (unless recycled).

Classification and clustering. Conversations could be seen as classes and the correlation problem could then be thought of as a classification one[1]. However, classification approaches assume a fixed (and rather small) number of classes while conversations come in unbounded and unknown number, depending mainly on the size of the log considered. Moreover, classification approaches rely on pre-classified instances to infer the classification

function. In our work, conversations to be used as training examples are not available. Note that inferring the correlation rules from a collection of conversation instances, if available, would be an interesting and complementary problem to explore.

One might also argue that correlation could be formulated as a clustering problem [9]. In clustering, the relationship between members of a cluster is their relative proximity according to some distance measure. However, messages of a same conversation may be very different (e.g. a purchase order and a payment message) while messages of two distinct conversation may be very similar (e.g. two purchase orders for the same products). Hence, clustering approaches—as well as other similarity-based approaches such as, e.g., Record Linkage[5]—could only be used provided a suitable and very ad-hoc distance measure has first been defined. Defining this measure is equivalent to identifying how to correlate messages; the purpose of this paper.

Session reconstruction. Web usage mining has raised the problem of *session reconstruction*. A session represents all the activities of a user in a Web site during a single visit. In this problem, however, the problem is not to identify how to correlate events in the log (this is typically done using the IP address) but how to decide on the boundaries of the session, i.e. after how much delay of inactivity would one consider that the same IP corresponds to a different user, or to the same user with discontinued activity. In many web services, using other types of information, e.g., identifiers, is more common for message correlation, or at least time is used in conjunction with other information. For instance, the WS-CDL standard proposal supports the usage of time with identifier-based correlation of messages in Web services.

Application dependency. Correlation is often cited in the context of dependency discovery, where the task is to identify whether some events may depend on some others. However, correlation in that context bears a different meaning than the one intended in this paper. It refers to a temporal dependency between events where, for example, an event is a cause that triggers one or more subsequent events. Examples of approaches in this category include several statistical approaches to numerical time series correlation [11] or event correlation for root cause analysis [19]). In this paper, correlation means grouping messages that belong to the same conversation. Although there might be a causal dependency between these messages (discussed hereafter), it is only incidental.

Message correlation in Web services. Correlation patterns in service workflows is studied in [3]. However, no automated support for message correlation is proposed. The work on service workflow discovery also characterizes the problem of time-based session reconstruction for Web services [4] and examines the possibility of adapting approaches of session reconstruction in Web usage mining to this problem. However, proposing an automated approach remains as future work. The need for automated approaches to

message correlation has first been recognized in [13] where a real situation on how to correlate messages is presented.

6 Conclusions and Future Work

In this paper, we have presented concepts, as well as a framework and a set of tools for discovering correlations in service logs, in order to reconstruct conversations. We introduced the notion of correlation dependency among messages in relational logs, where individual events correspond to tuples of a relation, and provided a framework to define, discover and heuristically explore possible correlation rules.

The framework can be extended in many directions. These extensions make it possible to adapt this correlation discovery framework in various contexts. We highlight below some future research directions.

Rule language. A lot of work has been done in the record linkage community to uncover relationships among seemingly separate entities. Users might be interested in discovering non-trivial correlation properties within collections of logs. The rule language used in this article is one of the most basic and can be extended in several directions to handle various situations. For instance, the equality between values can be replaced by a similarity function. Such rules would replace the current graph structure with a weighted graph and the problem of identifying disconnected components is replaced by that of identifying weakly connected components, i.e. components such that the aggregate of their relationship weights is below some threshold.

Heuristics. Depending on the context, heuristics can be modified to either enlarge or restrict the search space. For instance, in this paper, we limited the search space by taking into account the order relationship between log messages. In the context of web-services choreography, several logs may be involved and will first need to be integrated and then correlated. During log integration, it would be very difficult to keep a satisfying ordering of the messages since separate logs may have separate clocks. Richer relations can be used to replace $\prec$ and $\approx$ in order to reflect whether messages belong to the same or a different original log.

Time-based rule patterns. Note that discovering rule patterns based on the values of a time attribute is the subject of future work. One might need to define some functions over the time attributes of events in order to create log partitions. This way, events that share the same values of the attribute but are very far apart in terms of time (e.g., larger than a user provided threshold or a threshold discovered from the body of messages) cannot to be considered members of the same partition. Finally, one could also consider the partitioning of a log as a post-processing step, where events of each partition have a time difference larger than the average time difference of the events across all partitions of the log.

Imperfect data. Service log events may contain imperfections [14], e.g. incomplete conversations and noisy data in that there are missing events and/or event attribute values. Such imperfections may make the message correlation problem harder. We are planning to investigate extensions of our approach to tackle the problem of imperfect logs.

7 Acknowledgments

The authors would like to thank Pierre Buraud for his help in implementation of the approach presented in this paper.

References

[1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In L.-Y. Yuan, editor, *Proceedings of the 18th International Conference on Very Large Databases*, pages 560–573, San Francisco, U.S.A., 1992. Morgan Kaufmann Publishers.

[2] A. V. Aho, J. E. Hopcroft, J. Ullman, J. D. Ullman, and J. E. Hopcroft. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[3] A. Barros, G. Decker, M. Dumas, and F. Weber. Correlation patterns in service-oriented architectures. In *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE).* Springer Verlag, March 2007.

[4] S. Dustdar and R. Gombotz. Discovering web service workflows using web services interaction mining. *International Journal of Business Process Integration and Management (IJBPIM)*, Forthcoming.

[5] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[6] D. Grigori, F. Casati, U. Dayal, and M.-C. Shan. Improving business process quality through exception understanding, prediction, and prevention. In *The VLDB Journal*, pages 159–168, 2001.

[7] G. Group. Gartner exp report. In $www.gartner.com/press_releases/asset_143678_11.html$, 2006.

[8] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.

[9] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, 1988.

[10] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. In *ICDT '92: Selected papers of the fourth international conference on Database theory*, pages 129–149, Amsterdam, The Netherlands, The Netherlands, 1995. Elsevier Science Publishers B. V.

[11] H. Mannila and D. Rusakov. Decomposition of event sequences into independent components. In *Proc. of SIAM-SDM01*, January 2001.

[12] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Min. Knowl. Discov.*, 1(3):241–258, 1997.

[13] H. Motahari, B. Benatallah, and R. Saint-Paul. Protocol discovery from imperfect service interaction data. In *Proceedings of the VLDB 2006 Ph.D. Workshop*. ceur-ws.org/Vol-170, September 2006.

[14] 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, April 2007.

[15] W. Pauw and et. al. Web services navigator: Visualizing the execution of web services. *IBM System Journal*, 44(4):821–845, 2005.

[16] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 218–227, Washington, DC, USA, 1996. IEEE Computer Society.

[17] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. Gordian: Efficient and scalable discovery of composite keys. In *VLDB*, pages 691–702, 2006.

[18] M. Steinle, K. Aberer, S. Girdzijauskas, and C. Lovis. Mapping moving landscapes by mining mountains of logs: Novel techniques for dependency model generation. In *VLDB*, pages 1093–1102, 2006.

[19] M. Steinle, K. Aberer, S. Girdzijauskas, and C. Lovis. Mapping moving landscapes by mining mountains of logs: Novel techniques for dependency model generation. In *VLDB*, pages 1093–1102, 2006.

[20] W. van der Aalst and et. al. Workflow mining: a survey of issues and approaches. *DKE Journal*, 47(2):237–267, 2003.