
# Process Spaceship: Discovering Process Views in Process Spaces

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-0721
23 December 2007

This technical report builds on top of and extends the approach presented in the earlier technical report number UNSW-CSE-TR-0709, by the same authors, with the following details:

---

**Title**: Message Correlation for Conversation Reconstruction in Service Interaction Logs

**Authors**: Hamid R. Motahari Nezhad, Regis Saint-Paul, Boualem Benatallah, Fabio Casati and Periklis Andritso

**Publication Details**: Technical Report Number UNSW-CSE-TR-0709

The School of Computer Science and Engineering,

The University of New South Wales, Australia

**Date**: March 2007

**URL**: ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/0709.pdf

---

The material presented in this report has been submitted for publication, elsewhere.

**Abstract**

Process and service execution analysis is a key endeavour for enterprises. Such analysis requires observing and correlating messages related to process and service executions, meaning that identifying if messages belong to the same process instance or service execution. A first challenge is that message correlation is subjective, i.e., depends on the purpose of the analysis and on the perspective of the analyst. Another challenge lies in the huge space of possible correlations between messages, which can be built based on different combinations of message attributes.

In this paper, we consider process and service execution data as a *process space*, and different ways of performing correlations as *process views* that are views over the process space. We propose methods, by adopting a level-wise approach, and heuristics to identify the set of interesting process views and present a visual, interactive environment that allows users to efficiently navigate through the views identified over a process space. The experiments show the viability and efficiently of the approach on both synthetic and real-world service logs.

# 1 Introduction

The problem of understanding the behavior of information systems as well as 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 [17, 18, 27], and by recent research work in process data warehousing and process discovery [28, 8, 33]. Indeed, the adoption of business intelligence techniques for business process improvement is the primary concern for medium and large companies [14].

Typical questions that analysts, IT managers, developers and in some cases even users would like to have answered are: *where are the bottlenecks in the purchasing process? what is the actual process we typically follow for invoice payment? what is the status of purchase order number 325, who processed it and how?, and how to find all the information related to a specific purchase of a customer (order, invoice, payment, shipping, etc) in the enterprise?* (the latter is also referred to as enterprise search [15]).

Performing these kinds of analysis requires the ability to (i) *define* or *discover* a model of the process to be analyzed, used as a reference for asking queries such as those mentioned above (e.g., model of the purchase order process at the appropriate level of abstraction); ii) *map* events occurring in the various IT systems of the company to progressions of a process (i.e., to the start/completion of process tasks); (e.g., define that a data entry in a certain SAP table corresponds to the start of the supplier evaluation phase); and (iii) *correlate* events occurring in the different IT systems of the company to process instances, that is, to understand which events correspond to the same execution (instance) of a process (e.g., be able to detect that a data entry in SAP and a message sent over an enterprise service bus correspond to two events related to the same purchase order no. 325). Once these steps are done, we can interpret enterprise events in terms of executions of processes, and therefore we can perform process tracking and analysis.

The issue is analogous to that of *data spaces* [13], where the goal is to support data analysis (e.g., queries and joins) in environments characterized by heterogeneity in the information systems where data is kept, heterogeneity in the data formats, and lack of explicitly stated relationships that can identify correlation among such data elements. Here, we examine the information in the enterprise related to logical transactions and superimpose a process metaphor by identifying events, messages and information related to the same process execution, thereby going from a data space to a *process space*. The term "process space" refers to a set of events over which correlations, mapping, and process models have been defined, so that enterprise events can be interpreted in the context of executions of processes.

In a process space, different process models, mappings, and correlations, can be defined over the same set of events as different analysts may be interested in different views over such events (called *process views*). For example, the shipments of a set of goods may be related from the view of the warehouse manager, but if the goods are the results of different orders, they are unrelated from the view of the sales manager.

Tools that allow the definition of process views exist today and are widely used due to the ever increasing desire of accurate process tracking and analysis (e.g., HP OpenView [17]). They are successful, but they impose a heavy process modeling load and are error-prone. In addition, using these tools, defining correlations and mappings is difficult, if not impossible. More importantly, it is hard to manually maintain a process in the wake of changes in the enterprise IT [15].

Identifying correlations is a difficult task. Referring back to our purchase order (PO) example, we would be tempted to say that correlation should be based on an OrderID. However, it is unlikely that messages related to the same PO in different message logs, have a field called OrderID. Furthermore, the problem is often much

more complex than spotting some identifier in a set of log entries. In many cases, some messages related to the same PO processing may first be identified by a quotation number, then by the actual order number, and finally by the invoice number. The number of fields to be used as correlators and the number of possible combinations of such fields across log entries is potentially very high so that techniques and automated support for their identification is needed.

In this paper, we propose an approach, and a system called `Process Spaceship`, for the automated discovery of process views in process spaces. The problem of identifying correlations and deriving process views has two (related) sides: First, we need to analyze events and identify which events are correlated (belong to the same process execution). This is done based on discovering candidate correlation conditions that define which sets of events belong to the same execution trace of a process. In general, a condition partitions (a subset of) the events into execution traces of a process, and identifies a process view. Very often, there are many such possible correlation conditions (and sets of correlation conditions), so the search space and the number of possible views are very large. Hence, the second key task is to guide the analysts through the search space so that they can identify correlations and views they consider useful.

To facilitate this exploration, we process the execution traces and derive process models out of the traces, using a process discovery algorithm [33, 25]. This is very important as for analysts it is very hard to judge the usefulness of a view unless users can see the process model that underlies the view. In addition, we organize process views in a *process map* to help users in navigating among the possible views and to select the ones that are appropriate for their analysis purposes. A process map is essentially a graph that structures views based on their level of abstractions and the relationships of process models in process views.

In summary, this paper presents an approach to discover correlations conditions and process views over a data space. Specifically, we provide the following contributions:

- We introduce the notion of process space, characterize the problem of event correlation in process spaces, and introduce the notion of process views over a process space;
- We present techniques for the semi-automated identification of possible correlation conditions among messages in logs based on a number of heuristics that guide the search. We limit the user supervision to high level input that is reasonable to suppose known to them. Moreover, our process is designed to be robust with respect to the precision of the user supervision and can also perform without supervision at all.
- We propose to organize process views in a process map, and provide a visual, interactive interface for navigating through the process map. The navigation is based on the relationships that exist among process views. Process views of interest can be selected and used for the subsequent analysis;
- We substantiate the arguments by means of experiments, which show the viability and efficiency of the proposed approach on both synthetic and real-world datasets.

In this paper, we use as example problem the case of identifying process views by looking at the messages exchanged by one or more Web services. In particular, we are interested in identifying the processes collectively executed by services. Hence, we restrict the problem to the analysis of execution logs of services. This does not imply any loss of generality in the problem we are solving, with the exception that we do not deal in this paper with data space issues related to having heterogeneous data formats, but we "only" focus on correlation among events and on the identification of process views.

The rest of the paper is structured as follows: Section 2 characterizes the process views discovery problem. In Section 3 we present a framework and a set of algorithms for discovering process views. Section 4 presents the visual, interactive environment for process view exploration and refinement. In Section 5 we present implementation and experiments. We discuss related work in Section 6. Finally, we conclude and outline the future work in Section 7.

# 2 Problem Definition

In this section, we first characterize service interaction logs, the input, as well as the results we want to obtain, i.e., process views. We then explain the problem of producing the later from the former and define the concepts that will be used throughout the paper.

## 2.1 Service Interaction Logs

In Web services, an exchange of messages between two or more services is called a *conversation* (a conversation is a sequence of messages exchanged to fulfil a certain functionality, e.g., to order goods), and each conversation typically obeys some public process, also called *business protocol definition* [6, 8]. For example, various services and software components may exchange messages related to a purchase order, thereby taking part in a purchase order conversation, possibly performed following some purchase order protocol.

The events related to messages exchanged during service conversations can be logged using various infrastructures [11]. We use the terms "message" and "event" interchangeably in the following. For simplicity, we define a generic log model where each message $m$ is represented by a tuple of a relation $\mathcal{L} = \{m_1, m_2, \ldots, m_n\}$, and $m_i \in A_1 \times A_2 \times \cdots \times A_k$, where $k$ is the number of attributes and $n$ is the number of messages. Attributes $A_1, \ldots, A_k$ represent the union of all message attributes and each message typically contains only a subset of these attributes. We denote by $m_x.A_i$ the value of a message $m_x$ on attribute $A_i$. We further assume that each message $m_x$ has at least three other attributes: the timestamp at which the event is recorded ($m_x.\tau$), the sender of the message ($m_x.s$), and the receiver ($m_x.r$). Messages of a log are ordered by their timestamp. Figure 2(a) and 2(b) show two examples of message logs (where the timestamp attribute has been omitted). Web service interactions usually involve exchanging structured (XML) messages. A pre-processing ETL-like is required to extract features from XML documents and represent them as event tuples (see Section 5).

## 2.2 Process Views

A *process view* is a representation of the conversations that take place among services. Given a set of conversations, a process view offers various information regarding the corresponding process such as statistical information about, e.g., completion time of conversations as well as a graphical modeling of the process as, e.g., a state machines diagram or a Petri-net diagram. The graphical model is inferred using model discovery algorithms [33, 25]). In this paper, we will discuss the modeling of web services interactions and use the approach presented in [25] for discovering service protocols from collections of conversations. We give the detail of the information offered by process views in Section 4.

As mentioned in the introduction, there are often more than one way of grouping messages of a log into conversations which in turn lead to different process views. As

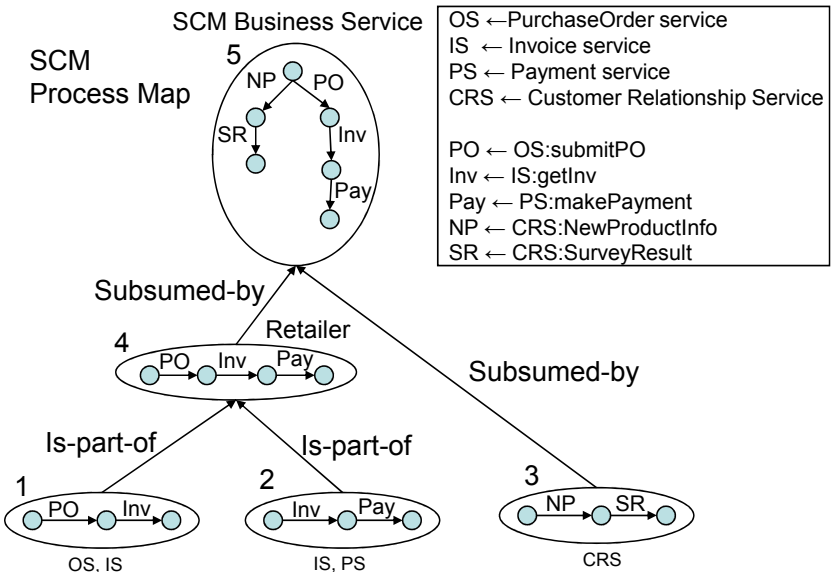



Figure 1: Part of process map for SCM dataset

an example, Figure 1 shows part of the process map discovered from the log of an SCM business service, in a supply chain management scenario, which is implemented by purchase order (OS), invoice (IS), payment (PS), and customer relationship (CRS) services. Symbol $PO$ denotes the short form of OS : $submitePO$ operation, i.e., the operation $submitPO$ of OS service. The extended form of other symbols ($Inv$, $Pay$, $NP$ and $SR$) are shown in Figure 1. The log contains the messages of interactions of SCM's customers with these Web services. The links between map nodes show the relationship of protocol models corresponding to each view (see Section 4).

The map is arranged according to the granularity of models and conversations, having small but highly correlated conversations (e.g., purchase order and invoice messages) at the lower levels, and large and possibly more loosely coupled conversations (e.g., including all messages related to PO and to its payment, or all messages related to the interaction with customer relationship service) at the higher levels. This organization makes it easy (or at least easier) to make sense of the various possible views and select those of interest based on the goal of the analysis and the interest of the team performing the analysis, as it identifies various protocols at different levels of abstraction and scope. Such an analysis also allows identifying how services in the enterprise depend on one another [28].

## 2.3 Grouping Messages into Conversations

Given a set of messages in a log $\mathcal{L}$, our objective is to find which messages correspond to the same service execution. For example, considering the SCM service log in Figure 1, we want to automatically discover that messages of $PO$ and $Inv$ belong to the same execution but that messages of type $NP$ do not. In most cases, this can be done by looking at the content of $PO$ and $Inv$ messages and observing that, e.g., they share the same purchase order ID. In this case, we say that messages $PO$ and $Inv$ are *correlated* and we call *correlation condition* the fact this stems from the two messages having a common value on attribute purchase order ID. In the rest of this section, we will formalize the notions of conversation and that of correlation condition in the context of Web services.

| | ConvID | CID | SID |
|---|---|---|---|
| $m_1$ | 1 | $c_1$ | $s_1$ |
| $m_2$ | 2 | $c_2$ | $s_1$ |
| $m_3$ | 1 | $c_1$ | $s_2$ |
| $m_4$ | 2 | $c_1$ | $s_1$ |
| $m_5$ | 2 | $c_2$ | $s_1$ |
| $m_6$ | 1 | $c_1$ | $s_2$ |

(a)

| | oID | OrdRef |
|---|---|---|
| $m_1$ | 001 | |
| $m_2$ | 002 | |
| $m_3$ | 003 | 001 |
| $m_4$ | 004 | 002 |
| $m_5$ | 005 | 003 |
| $m_6$ | 006 | 005 |

(b)

Figure 2: Snapshots of service logs

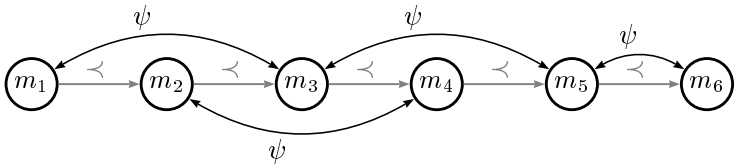



Figure 3: Correlation graph $G_\psi(\mathcal{L})$ for log in Figure 2(b) ($\prec$ shows time precedence)

### 2.3.1 From messages to conversations

We assume that messages in the log contain the information needed to correlate them. This assumption is reasonable since messages are indeed correlated by the recipient services. Based on this assumption, a correlation condition can be defined as follows:

**Definition 1** *A correlation condition is a binary predicate defined over attribute values of two messages $m_x$ and $m_y$ and denoted by $\psi(m_x, m_y)$. This predicate is true when $m_x$ and $m_y$ are correlated and false otherwise.*

For example, in the log presented in Figure 2(a), we can have the correlation condition $\psi(m_x, m_y) : m_x.\mathsf{CID} = m_y.\mathsf{CID}$. A correlation condition $\psi$ allows to partition a log of messages $\mathcal{L}$ into a set of conversations based on the following definition:

**Definition 2** *A conversation $c$ is a sequence of messages $c = \langle m_1, m_2, \dots \rangle$ corresponding to a subset of messages of the log $\mathcal{L}$, ordered by their timestamp, and such that (i) any message $m_x \in c$ is correlated with at least one other message $m_y \in c, y \neq x$ and (ii) all the messages $m \in \mathcal{L}$ correlated with at least one message of $c$ are also in $c$ (i.e., $c$ is a maximal subset with respect to the correlation condition).*

To better see how a correlation condition $\psi$ partitions the log into conversations, let us represent the relationships between messages in log $\mathcal{L}$ using a graph $G_\psi(\mathcal{L})$, in which the set of vertices corresponds to the set of messages $\mathcal{L}$ and an edge is placed between two messages $m_x$ and $m_y$ if $\psi(m_x, m_y)$ holds. We denote $R_\psi$ the set of edges in $G_\psi$ (correlated message pairs based on $\psi$), i.e., we have $R_\psi = \{(m_x, m_y) \in \mathcal{L}^2 | \psi(m_x, m_y)\}$. For example, the correlation graph corresponding to the log in Figure 2(b) with $\psi : m_x.\mathsf{oID} = m_y.\mathsf{OrdRef}$ is illustrated in Figure 3. Conversations correspond to connected components, i.e., maximal subgraphs of $G_\psi(\mathcal{L})$, and the partitioning of the log $\mathcal{L}$ by a correlation condition $\psi$ corresponds to the partitioning of the correlation graph $G_\psi(\mathcal{L})$ into connected components. We denote by $C_\psi(\mathcal{L})$ the set of conversations that results from partitioning $\mathcal{L}$ using the correlation condition $\psi$. For example, with $\mathcal{L}$ the log presented in Figure 2(b), we have $C_\psi(\mathcal{L}) = \{\langle m_1, m_3, m_5, m_6 \rangle, \langle m_2, m_4 \rangle\}$.

In the following, we discuss the different approaches used to correlate web service messages in real world service interactions. From these considerations, we will identify classes of correlation conditions that need to be investigated when trying to correlate messages in the log to form conversations.

### 2.3.2 Correlation conditions in service-based processes

Web services standard proposals, such as BPEL, WS-Conversation, and WS-CDL, as well as industrial software products (e.g., IBM Websphere Process Manager) for process definition and monitoring of Web services provide a variety of mechanisms for message correlation. The most common approach for correlating messages in Web services is through having some identifier(s) shared by messages of the same conversation [3].

In some cases, the identifier value used for correlating messages may be unique for all messages of a conversation. This value acts as a "key" that uniquely identifies a conversation. For instance, the attribute ConvID in the log of Figure 2(a) acts as a key since messages $m_1$, $m_3$ and $m_6$ all have the value "1" on this attribute. The corresponding correlation condition is $\psi_{key} : m_x.\mathsf{ConvID} = m_y.\mathsf{ConvID}$, and we have $C_{\psi_{key}}(\mathcal{L}) = \{\langle m_1, m_3, m_6\rangle, \langle m_2, m_4, m_5\rangle\}$. This method of correlation is called a *key-based correlation*.

In other cases, messages of conversations are correlated using a reference to a previous message in the conversation. In this case, in any message (except the first) there is a reference attribute sharing a unique value with an identifier attribute in a previous message. For example, messages in Figure 2(b) are correlated using this method, called *reference-based correlation*, by the correlation condition $\psi_{ref} : m_x.\mathsf{OID} = m_y.\mathsf{OrdRef}$. In this method, messages of the same conversation form a chain as illustrated in Figure 3. This is also referred to as conversations with chain-based correlation.

The two correlation methods described above can be modeled using a first class of correlation condition—called *atomic conditions*—defined as an equality relationship over attributes of two messages as follows:

**Definition 3** *An atomic condition $\psi$ specifies that two messages are correlated if they have the same value on two of their attributes $A_i$ and $A_j$, that is, $\psi : m_x.A_i = m_y.A_j$.*

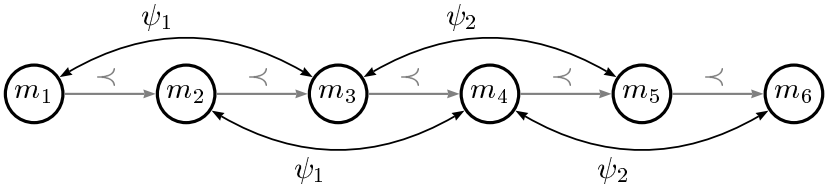
Similarly to the concept of composite keys in databases, where keys may consist of more than one attributes, the method used for correlating messages may use several attributes. For instance, messages of a conversation may be correlated using the values of attributes customer ID ($\psi_1 : m_x.\mathsf{CID} = m_y.\mathsf{CID}$) and survey ID ($\psi_2 : m_x.\mathsf{SID} = m_y.\mathsf{SID}$), as for messages $NP$ and $SR$ (Figure 1, node 3). Figure 2(a) shows a log corresponding to this scenario (assuming attribute ConvID is not present). In this case, the correlation condition can be defined as $\psi_c = \psi_{1\wedge 2} = \psi_1 \wedge \psi_2$. We refer to this second class of correlation condition as *composite conjunctive* (for short, *conjunctive*). In this example, $\psi_{1\wedge 2}$ partitions the log into three conversations, i.e., $C_{\psi_{1\wedge 2}}(\mathcal{L}) = \{\langle m_1, m_4\rangle, \langle m_2, m_5\rangle, \langle m_3, m_6\rangle\}$.

**Definition 4** *A composite conjunctive correlation condition is a conjunction of more than one atomic conditions. It follows the general form of $\psi : \psi_1 \wedge \psi_2 \wedge \ldots$ where $\psi_1$, $\psi_2$, $\ldots$ are atomic conditions.*

In another scenario, several correlation conditions may be needed to correlate the messages of the same conversation. For example, consider the log in Figure 4(a) (the corresponding protocol is illustrated in node 4, Figure 1). Messages of type $PO$ and $Inv$ are correlated using the condition $\psi_1 : m_x.\mathsf{oID} = m_y.\mathsf{oID}$ and messages of type $Inv$ and $Pay$ are correlated using the condition $\psi_2 : m_x.\mathsf{invID} = m_y.\mathsf{invID}$ but all are part of the same conversation. For such conversations, messages $m_x$ and $m_y$ are correlated if they satisfy *either* $\psi_1$ *or* $\psi_2$ (or they may satisfy both). In Figure 4(b) messages $m_1$ and $m_3$ (resp. $m_3$ and $m_5$) are correlated according to $\psi_1$ (resp. $\psi_2$). These three messages form a unique conversation under condition $\psi_d = \psi_{1\vee 2} = \psi_1 \vee \psi_2$. In the case of node 4 of Figure 1, messages of same conversations also form a chain (see Figure 4(b)). We refer to such cases, also, as conversation with chain-based correlation.

| name | | oID | invID | pID |
|---|---|---|---|---|
| $PO$ | $m_1$ | $o_1$ | | |
| $PO$ | $m_2$ | $o_2$ | | |
| $Inv$ | $m_3$ | $o_1$ | $i_1$ | |
| $Inv$ | $m_4$ | $o_2$ | $i_2$ | |
| $Pay$ | $m_5$ | | $i_1$ | $p_1$ |
| $Pay$ | $m_6$ | | $i_2$ | $p_2$ |

(a)




(b) $\psi_1 : m_x.\mathsf{OID} = m_y.\mathsf{OID}$, $\psi_2 : m_x.\mathsf{invID} = m_y.\mathsf{invID}$

Figure 4: (a) snapshot of the log for node 4 in Figure 1, (b) corresponding correlation graph $G_{\psi_1 \vee \psi_2}$

Finally, a protocol may accept conversations where each one of them is correlated using different correlation conditions. For instance, the protocol illustrated in node 5 of Figure 1 allows `CRM` conversations (node 3)—correlated using condition $\psi_c$—as well as purchase conversations (node 4), correlated with $\psi_d$ (see above). For producing the complete set of conversations corresponding to node 5, we have to group messages by either $\psi_c$ or $\psi_d$. This means that it can be achieved using the disjunctive condition $\psi_c \vee \psi_d$ as the set of conversations of node 5 is the union of the set of those of nodes 3 and 4.

**Definition 5** *A composite disjunctive correlation condition is a disjunction of more than one atomic or conjunctive conditions. It follows the general form* $\psi : \psi_1 \vee \psi_2 \vee \ldots$ *where* $\psi_i s$ *are either atomic or conjunctive conditions.*

*Time constraints on correlation conditions.* In some applications, time constraints are also used as part of correlation condition definitions. For example, in addition to equality relationships, WS-CDL allows to define a time limit for a conversation. In terms of correlation condition, this would mean that two messages are correlated only if their time difference is below some maximal duration $MaxDuration$, i.e., $\psi : m_x.A_i = m_y.A_j \wedge |m_x.\tau - m_y.\tau| \leq MaxDuration$.

## 2.4 Problem Statement

The problem we address in this paper is that of automatically inferring a process map, i.e., an organized visualization of all the process views relevant to a log. We have seen that each process view corresponds to a set of conversations obtained by using a correlation condition to correlate messages in the log. This problem can then be formulated as an exploration of the possible correlation conditions. This exploration is difficult as the number of potential correlation condition, if considered from a purely combinatorial point of view, is huge. One might first try each atomic condition, that is, the $a = k^2/2$ possible pairs of attributes if the log has $k$ attributes. Then, one might attempt to combine these atomic conditions to form conjunctive and disjunctive composite conditions. In theory, there can be $c = 2^a - 1$ conjunctive conditions to explore, and finally $d = 2^{(a+c)} - 1$ disjunctive conditions.

An exhaustive search would not scale. Moreover, many of the correlation conditions produced are not interesting to the user. For instance, grouping messages based on the total amount of a purchase is unlikely to produce an interesting process view. The main challenges thus for the automated approach are:

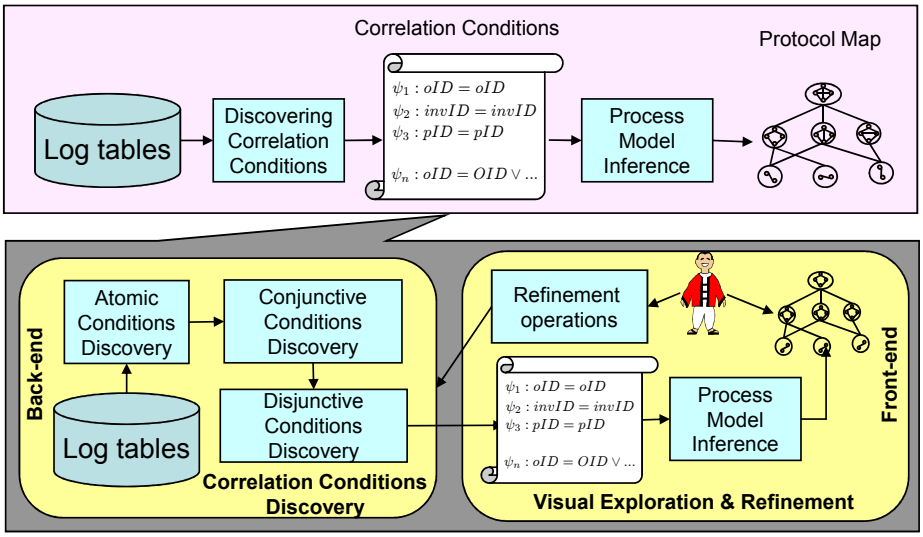



Figure 5: Architecture of `Process Spaceship`, a visual, interactive framework for process space discovery

- How to define interestingness of process views based on some objective criteria, given it is subjective?
- How to efficiently search the space of possible conditions and to avoid unnecessary computations, e.g., by exploiting the properties of conversation sets (e.g., inclusion of a set into another)?
- How to incorporate user domain knowledge to further focus the condition space exploration?
- Once discovered, how to help the user in understanding and navigating through process views?

In the next section, we will detail our framework for process view discovery and how each of the aforementioned points is addressed.

# 3 Discovery of Process Views

We present a framework, a system based on that called `Process Spaceship`, and a set of algorithms for discovery of process views to address the challenges outlined in Section 2.4. Figure 5 depicts its architecture, organized into two components: a back-end, responsible for discovering interesting views (presented in the following of this section), and a front-end, for visualization and user-driven refinement (see Section 4 and Section 5.3).

The problem of discovering process views is recasted as a problem of discovering correlation conditions from the log, coupled with the application of a model discovery algorithm to derive process models, and helping users to navigate the map of models. To explore the space of possible conditions efficiently, we adopt a level-wise approach [23] and use a set of criteria to reduce the space of possible correlation conditions. In this approach, the set of candidate conditions is grown from atomic to composite (conjunctive and disjunctive) and, at each level process views that do not satisfy objective interestingness measures are pruned. In the following, we identify interestingness measures for correlation conditions.

## 3.1 Interestingness of Correlation Conditions

While interestingness is subjective, it is possible to identify interesting of results based on identifying what is not interesting, for which we can define objective measures [29].

### 3.1.1 Non-Interestingness criteria

A correlation condition is interesting if it leads to an interesting process view. Thus, an interesting correlation condition should lead to forming a set of conversations in the log. We use heuristics on (i) the properties of attributes that can form interesting conditions, and on, (ii) how an interesting condition should partition the log into conversations, as a means to define non-interestingness criteria:

**(A) Attributes with non-repeating values are not interesting**: Correlation is based on equality of values in two or more messages. The values of correlator attributes are repeated either in the same attribute (key-based correlation) or in different attributes (reference-based correlation). Attributes having unique values, i.e., their values not found in any attributes of other tuples, can be tagged as non-interesting. Conversely, attributes with very small domains, i.e., their values repeat a lot, are not interesting either as they lead to few trivial partitions. To characterize these properties, we define the following two measures:

- $distinct\_ratio(A_i)$: for key-based conditions on attribute $A_i$, this ratio corresponds to the number of distinct values for an attribute $A_i$ w.r.t. the number of non-null values in $A_i$;
- $shared\_ratio(\psi_{ij})$: for reference-based conditions between attributes $A_i$ and $A_j$, this ratio represents the number of shared values of attributes $A_i$ and $A_j$ w.r.t. the number of non-null values in the two attributes.

Moreover, categorical attributes (e.g., those contain error codes or currencies) which are not used for correlation can be characterized in the log by the fact the number of values does not vary much with respect to the size of the dataset. Conversely, an attribute used for correlation would have more distinct values as the dataset grows since the dataset would contain more conversations. We use this property to further filter out non-correlator attributes, by comparing their value distribution on samples of the dataset of varying sizes. Thus, if the highest number of distinct value for a categorical attribute identified in this approach is denoted by $distinct_{max}(A_i)$, we use a threshold $\alpha = |distinct_{max}(A_i)|/|\mathcal{L}|$ and prune key-based conditions with the ratio of distinct values smaller than $\alpha$ (usually $\alpha \leq 0.01$). Based on a similar reasoning, we can prune reference-based conditions with $shared\_ratio(\psi_{ij}) < \alpha$. Finally, key-based conditions with $distinct\_ratio(A_i) = 1$ are also considered non-interesting. This is because using a key-based condition based on such attributes no messages can be correlated to each other.

**(B) Conditions partitioning the log into conversations of length less than 2, or into very few conversations, are not interesting**: A correlation condition $\psi$ is not interesting if it partitions the log into either very few conversations (i.e. all messages are correlated to each other) or into a very high number of conversations (i.e., correlates too few messages). In the latter case, many conversations are of length 1 (isolated messages). Hence, we can filter such conditions, e.g., by examining the median of the distribution of length of conversations and tag as non interesting those with median equal to 1. Note that for forming conversations, only messages in the log are considered where attributes $A_i$ and $A_j$ of a conversation $\psi_{ij}$ have non-null values.

To identify non-interesting condition of the former category (very few conversations), we define $Conv\_ratio$ as the ratio of $|C_\psi(\mathcal{L})|$ to the number of messages for which attributes used in the condition $\psi$ are defined (i.e., they are not null). For instance, for $\psi_1$ in Figure 4, only messages of types $PO$ and $Inv$ have non-null values for the attribute $OID$, i.e., there are 4 messages having non-null values. We require that

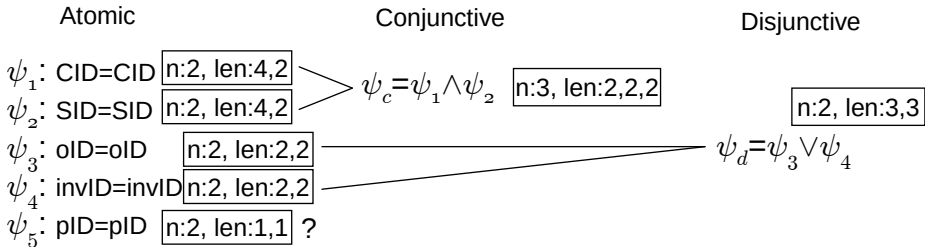



Figure 6: Condition Discovery for `Retailer` dataset

the majority of conversations have at least a length of 2 and therefore, $Conv\_ratio$ can be expected to be greater than 0.5. Therefore, we can safely tag correlation conditions that lead to a $Conv\_ratio$ below this threshold as non-interesting.

### 3.1.2 User domain knowledge

Users may have some knowledge regarding the processes they want to analyze. This knowledge can be leveraged to help further guide the search and enhance the quality of the discovery by refining the collection of views presented to the user. For instance, users may be able to give an estimation of the number, length and/or duration (i.e., time elapsed between the first and last message) of conversations. For example, a salesman may know that, in average, 100 orders are filed daily by customers.

We allow users to specify any of the following three interdependent criteria: the average number of conversations, their average length or their average duration. Note that we do not expect the user to provide a precise value for these metrics. They are used as indication to refine the analysis. We show how this knowledge, if available, can be used during the discovery process, and also after discovering views to effectively navigate through them.

## 3.2 Discovery of Correlation Conditions

We propose to discover the space of correlation conditions by discovering first atomic conditions, then conjunctive conditions (formed by combining atomic conditions), and finally disjunctive conditions (formed by combining atomic and conjunctive conditions). Figure 6 illustrates how these three steps are applied on the log presented in Figure 2(a) (excluding convID attribute) and Figure 4(a), which correspond to nodes 3 and 4 in Figure 1, respectively. In this figure, $n$ is used to denote the number of conversations that are result of applying this condition on the log, and $len$ gives the distribution of length of respective conversations. This example is used for illustration throughout this section.

### 3.2.1 Discovering atomic conditions

The approach for discovering atomic conditions is depicted in Algorithm 1. The algorithm consists of three steps:

**Generating candidate atomic conditions** (line 1). In this step, from the set of attributes in $\mathcal{L}$, we first generate the set of candidate atomic conditions $\psi_{ij} : m_x.A_i = m_y.A_j$, e.g., $m_x.\mathsf{CID} = m_y.\mathsf{CID}$;

**Pruning non-interesting conditions based on criterion (A)** (lines 2 to 9). In this step, first, key-based conditions are identified and pruned (lines 2 to 5). For this purpose, $distinct\_ratio(A_i)$ is computed for all attributes participating in a key-based

**Algorithm 1** Generation and pruning of atomic conditions

**Require:** $A$: the set of attribute $A_i \in \mathcal{L}, 1 \leq i \leq k$
**Ensure:** $AC$: the set of atomic conditions
1: $AC \leftarrow$ the set of conditions $\psi_{ij} : m_x.A_i = m_y.A_j$
2: **for** conditions $\psi_{ii} : m_x.A_i = m_y.A_i$ **do**
3: $\quad distinct\_ratio(A_i) = \frac{distinct(A_i)}{nonNull(A_i)}$
4: **end for**
5: $AC \leftarrow AC$ - $\{\psi_{ii} | distinct\_ratio(A_i) < \alpha$ or $distinct\_ratio(A_i) = 1\}$
6: **for** conditions $\psi_{ij} : m_x.A_i = m_x.A_j, i \neq j$ **do**
7: $\quad shared\_ratio(\psi_{ij}) = \frac{|distinct(A_i) \cap distinct(A_j)|}{max(|distinct(A_i)|, |distinct(A_j)|)}$
8: **end for**
9: $AC \leftarrow AC$ - $\{\psi_{ij} | shared\_ratio(\psi_{ij}) < \alpha\}$
10: **for** all conditions $\psi \in AC$ **do**
11: $\quad$ compute $R_\psi$
12: $\quad C_\psi(\mathcal{L}) \leftarrow FindConversations(R_\psi, \mathcal{L})$
13: **end for**
14: **if** $\psi$ not interesting according to criteria (B) on $C_\psi(\mathcal{L})$ **then**
15: $\quad AC \leftarrow AC - \{\psi\}$
16: **end if**

condition, and then criterion (A) is applied. For example, in Figure 6, condition $\psi_5$ is pruned as it has all unique values ($distinct\_ratio(pID) = 1$). Next, non-interesting reference-based conditions are identified and pruned (lines 6 to 9). For this purpose, $shared\_ration(\psi_{ij})$ is computed for attribute pairs participating in reference-based conditions and then criterion (A) is applied.

**Pruning non-interesting conditions based on criterion (B)** (lines 10 to 16). This step requires: (i) computing the set of correlated message pairs ($R_\psi$) for all conditions $\psi$ (line 11), (ii) computing the set of conversation $C_\psi(\mathcal{L})$ formed by correlated message pairs in $R_\psi$ (line 12), (iii) applying criterion (B) on $C_\psi(\mathcal{L})$ (lines 14 to 16). These sub-steps are explained in the followings.

*Computing the set of correlated message pairs for a condition $\psi$ .* The set of correlated messages pairs $R_\psi$ can be computed using a standard SQL query (self-join of log $\mathcal{L}$) in which its `WHERE` clause applies the condition $\psi$, e.g.:

```
SELECT a.id, b.id
FROM   ℒ a, ℒ b
WHERE  a.Ai=b.Aj AND b.id > a.id
```

Here, it is assumed that there is a message identifier ($id$) that can be compared to each other. The condition $b.id > a.id$ makes sure that only the correlation between each message and the following messages in the log $\mathcal{L}$ are considered. This is because each message can correlated to messages arrived after itself in the log. Given that the correlation between messages is undirected, it is enough to look forward from each message. Doing so, we make sure that the relationships between any previous message and the current one is identified from the previous message. This also allows to save a lot in the memory used, also the number of tuples in the database, to store and keep $R_\psi$.

*Computing the set of conversations for a condition $\psi$ .* The computation of the set of conversations $C_\psi(\mathcal{L})$ from $R_\psi$ corresponds, in terms of database query, to a recursive (closure) query over the set of correlated message pairs in $R_\psi$. In fact, a conversation is the transitive closure of binary relationships between messages in $R_\psi$. Because the support for recursive queries is limited in current database engines, we choose to use the correlation graph $G_\psi(\mathcal{L})$ represented by $R_\psi$ and partition it by applying a graph decomposition algorithm using off-the-shelf algorithms for this

purpose [9]. In particular, from implementation point of view, this is done in three steps: (i) computing correlation message pairs for $\psi$ ($R_\psi$), (ii) building the graph $G_\psi(\mathcal{L})$ from $R_\psi$, and (iii) applying the decomposition algorithm on $G_\psi(\mathcal{L})$. More details about this is presented in Section 5.

### 3.2.2 Discovering composite conjunctive conditions

Conjunctive conditions $\psi_{1\wedge 2}$ are computed using conjunctive operator on atomic conditions $\psi_1$ and $\psi_2$, i.e., $\psi_{1\wedge 2}$=$\psi_2 \wedge \psi_2$. If the set of atomic correlation conditions computed in the previous step is $AC = \{\psi_1, \psi_2, \psi_3\}$, then the set of possible candidate conjunctive condition is $CC = \{(\psi_1 \wedge \psi_2), (\psi_1 \wedge \psi_3), (\psi_2 \wedge \psi_3), (\psi_1 \wedge \psi_2 \wedge \psi_3)\}$. This corresponds to exploring the set containment lattice of $AC$.

It is possible that some of these combinations, built using more atomic conditions from simpler ones (with fewer atomic conditions), lead to the same set of conversations as that of the simpler ones. In such cases, it is enough to find only the minimal conjunctive conditions. A conjunctive condition $\psi$ is minimal if no other conjunctive condition formed using fewer conjunction of atomic conditions partitions the log into the same set of conversations. For example, assume that, in the set $AC$ above, the conjunctive conditions $\psi_{1\wedge 2}$ and $\psi_{1\wedge 2\wedge 3}$ partition the log in the same set of conversations, then $\psi_{1\wedge 2\wedge 3}$ is not minimal and $\psi_{1\wedge 2}$ is desired as it is easier to compute.

Hence, there are two requirements for an automated approach to discover conjunctive conditions: (i) efficiently explore the set containment lattice of atomic conditions, by discovering only interesting conditions, and (ii) discover only minimal conjunctive conditions. To fulfil these requirements, in the following we propose an algorithm by adopting a level-wise iterative approach [23]. At each level $L_i$, more complex conjunctive conditions (i.e., formed using more atomic conditions) are grown from simpler conditions (i.e., formed from fewer atomic conditions) of the previous level $L_{i-1}$.

The proposed algorithm is depicted in Algorithm 2. Each iteration of the algorithm has three phases: applying conditions $\psi$ to partition the log into conversations (lines 4 to 7), candidate condition pruning (lines 8 to 10), and generation of candidate conditions for the next level (lines 12 to 17). The algorithm ensures that only minimal conjunctive conditions are discovered, as explained in the following.

**Partitioning the log into conversations for a conjunctive condition $\psi_{1\wedge 2}$.** For a candidate conjunctive condition $\psi_{1\wedge 2}$, the first step is to compute the set of correlated messages pairs $R_{\psi 1\wedge 2}$. This is defined as the intersection of the correlated messages pairs of $\psi_1$ and $\psi_2$ as follows:

$$\begin{aligned}(m_x, m_y) \in R_{\psi 1\wedge 2} \quad &\Leftrightarrow \quad (m_x, m_y) \in R_{\psi 1} \wedge (m_x, m_y) \in R_{\psi 2} \\ &\Leftrightarrow \quad (m_x, m_y) \in R_{\psi 1} \cap R_{\psi 2}\end{aligned}$$

this means that messages $m_x$ and $m_y$ have same values for attribute pairs in $\psi_1$ *and* $\psi_2$. $R_{1\wedge 2}$ can be computed as an SQL query using the `INTERSECT` operator over $R_{\psi 1}$ and $R_{\psi 2}$. Computation of $C_{\psi 1\wedge 2}(\mathcal{L})$ can be done based on finding connected components of its correlation graph $G_{\psi 1\wedge 2}(\mathcal{L})$, as explained in Section 3.2.1.

**Pruning candidate conjunctive conditions**. This phase, non-interesting conjunctive correlation conditions are identified and pruned based on the following criteria.

(1) *Criterion (B) in Section 3.1.1 is applied to identify and prune non-interesting conjunctive conditions* (lines 8 to 10 of Algorithm 2). In particular, the number of conversations for $\psi_{1\wedge 2}$ is necessarily equal or greater than that of both $\psi_1$ and $\psi_2$ (e.g., consider condition $\psi_c$ in Figure 6). Therefore, we check if the condition $Conv\_ratio(\psi_{1\wedge 2}, \mathcal{L}) < \beta$ is still satisfied (i.e., if the conjunctive

**Algorithm 2** Generation and pruning of conjunctive conditions

**Require:** $AC$: the set atomic conditions
**Ensure:** $CC$: the set of atomic and conjunctive conditions
1: $L_0 \leftarrow \{\}$; $L_1 \leftarrow AC$
2: $\ell \leftarrow 1$
3: **while** $L_\ell <> \{\}$ **do**
4: **for** condition $\psi \in L_\ell$ **do**
5: compute $R_{\psi 1 \wedge 2} \leftarrow R_{\psi 1} \cap R_{\psi 2}$
6: $C_\psi(\mathcal{L}) \leftarrow FindConversations(R_\psi, \mathcal{L})$
7: **end for**
8: **if** not interesting according to criteria (B) on $C_\psi(\mathcal{L})$ *or* $notMon(\psi_{1\wedge 2})$ **then**
9: $L_\ell \leftarrow L_\ell - \{\psi\}$
10: **end if**
11: $CC \leftarrow CC \cup L_\ell$
12: **for** conditions $\psi_1, \psi_2 \in L_\ell$ **do**
13: $\psi_{1\wedge 2} \leftarrow \psi_1 \wedge \psi_2$
14: **if** $def(\psi_{1\wedge 2})$ *or* $notInc(R_{\psi 1}, R_{\psi 2})$ **then**
15: $L_{\ell+1} \leftarrow L_{\ell+1} \cup \{\psi_{1\wedge 2}\}$
16: **end if**
17: **end for**
18: $\ell \leftarrow \ell + 1$
19: **end while**

condition is potentially interesting). If the user knowledge is provided, e.g., on the number and length of conversations, in this step, it can be used to prune non-interesting conditions, as well.

(2) *Monotonocity of the number and the length of conversations with respect to the conjunctive operator*: As mentioned before, we expect that the number of conversations for $\psi_{1\wedge 2}$ is greater than that of both $\psi_1$ and $\psi_2$. This also implies that (most of) conversations $C_{\psi 1\wedge 2}(\mathcal{L})$ are of smaller length than those of $\psi_1$ and $\psi_2$. Therefore, if the number of conversations does not increase, or the length of (at least some of) conversations in $C_{\psi 1\wedge 2}(\mathcal{L})$ is not smaller than those of $C_{\psi 1}(\mathcal{L})$ and $C_{\psi 2}(\mathcal{L})$, then $\psi_{1\wedge 2}$ is not interesting. This is because such view does not create a new interesting process view. The slight change in the number or the length of conversation (since $C_{\psi 1\wedge 2}(\mathcal{L}) \neq \psi_1$ and $C_{\psi 1\wedge 2}(\mathcal{L}) \neq \psi_2$) are due to some imperfection in the log. We apply this condition to prune non-interesting conditions. This criterion is referred to as $notMon(\psi_{1\wedge 2})$ in line 8 of Algorithm 2.

**Generating candidate conjunctive conditions**. In this phase the set of candidate conditions for the next level is generated (lines 12 to 17). The set of candidate conditions for the next level are formed using non-pruned (selected) correlation conditions of the previous level. In fact, if a condition $\psi_1$ is pruned, i.e., it fails to satisfy the interestingness measures (e.g., the number of conversation is too high or conversations are too short), then the conjunctive condition built using this and any other conditions will also fail to satisfy these criteria since the resulting conversations will necessarily be shorter or of the same length at best. Therefore, only selected conditions from the previous level are used.

We use the following criteria to predict that some candidate condition are non-interesting without having to compute how they partition the log, which is computationally expensive (see Section 3.3 and Section 5):

(1) *Attribute definition constraint*: Consider two atomic conditions $\psi_1$ and $\psi_2$. Each of the attributes used in $\psi_1$ (e.g., $A_{i_1}$ and $A_{j_1}$) and $\psi_2$ (e.g., $A_{i_2}$ and

$A_{j_2}$) may be undefined for some messages (consider $\psi_1$ and $\psi_2$ in Figure 4). However, when considered together in a conjunction, a new constraint appears: attributes of $\psi_2$ have to be defined whenever the attributes of $\psi_1$ are defined. More formally, if we denote $\psi_{1\wedge 2}$ the condition formed by the conjunction of $\psi_1$ and $\psi_2$, it has the following form:

$$\psi_{1\wedge 2} : m_x.A_{i_1} = m_y.A_{j_1} \wedge m_x.A_{i_2} = m_y.A_{j_2}.$$

Hence, for any message of the log, attribute $A_{i_1}$ (resp. $A_{j_1}$) is defined if and only if $A_{i_2}$ (resp. $A_{j_2}$) is also defined. Therefore, we can verify if $\psi_1$ and $\psi_2$ have definite values for the same set of messages. If not, this conjunction can be safely discarded and we can avoid computing its corresponding log partitioning into conversations. This criterion is referred to as $def(\psi_{1\wedge 2})$ in line 14 of Algorithm 2.

(2) *Inclusion Property.* If the set of messages correlated by $\psi_1$ is included in that of $\psi_2$ (i.e., if we have $R_{\psi_1} \subseteq R_{\psi_2}$), then we have $R_{\psi_1\wedge\psi_2} = R_{\psi_1}$. Therefore, $\psi_1 \wedge \psi_2$ is not minimal (since it builds down to $\psi_1$) and we can avoid computing its conversations. Furthermore, if we have $R_{\psi_1} = R_{\psi_2}$, then $\psi_1$ and $\psi_2$ partition the log in the same way (they produce the same view), and it is enough to consider only one of them in all later computations. This criterion is referred to as $notInc(R_{\psi 1}, R_{\psi 2})$ in Algorithm 2.

### 3.2.3 Discovering composite disjunctive conditions

Similar to discovering conjunctive conditions, discovering disjunctive conditions consists in finding the set of interesting minimal disjunctive combinations of candidate correlation conditions. For example, if the set of correlation conditions (atomic or conjunctive) computed in the previous steps is $CC = \{\psi_1, \psi_2, \psi_3\}$, then the set of possible candidate disjunctive condition is $MC = \{(\psi_1 \vee \psi_2), (\psi_1 \vee \psi_3), (\psi_2 \vee \psi_3), (\psi_1 \vee \psi_2 \vee \psi_3)\}$. This corresponds to exploring the set containment lattice of $CC$.

We also propose an algorithm by adopting a level-wise approach [23] to search the space of possible disjunctive conditions, similar to Algorithm 2 for discovery of conjunctive conditions. This is performed in an iterative process comprised of three phases: finding conversations, candidate pruning, and next level candidate generation. The respective algorithm is depicted in Algorithm 3. The input of this algorithm is $CC$, which contains both atomic and conjunctive conditions. It ensures that all minimal disjunctive conditions are discovered.

**Finding conversations for disjunctive condition $\psi_{1\vee 2}$.** For a disjunctive condition $\psi_{1\vee 2}$, we have $R_{\psi 1\vee 2} = R_{\psi 1} \cup R_{\psi 1}$, i.e., the set of correlated message pairs of $\psi_{1\vee 2}$ is the union of set of message pairs $\psi_1$ and those of $\psi_2$:

$$\begin{aligned}(m_x, m_y) \in R_{\psi 1\vee 2} \quad &\Leftrightarrow \quad (m_x, m_y) \in R_{\psi 1} \vee (m_x, m_y) \in R_{\psi 2} \\ &\Leftrightarrow \quad (m_x, m_y) \in R_{\psi 1} \cup R_{\psi 2}\end{aligned}$$

$R_{\psi 1\vee 2}$ is computed using an SQL query based on `UNION` operator on $R_\psi 1$ and $R_\psi 2$ (step 5). Then, the set of conversations $C_{\psi 1\vee 2}(\mathcal{L})$ is computed based on finding connected components of its correlation graph $G_{\psi 1\vee 2}(\mathcal{L})$ using a graph decomposition algorithm (step 6), as discussed in Section 3.2.1.

**Pruning candidate disjunctive conditions**. In this phase, the following criteria are used to prune non-interesting correlation conditions:

(1) *Criterion (B) in Section 3.1.1 is applied to identify and prune non-interesting disjunctive conditions.* Conversations formed based on disjunction

**Algorithm 3** Generation and pruning of disjunctive conditions

**Require:** $CC$: the set atomic and conjunctive conditions
**Ensure:** $CC$: the set of atomic, conjunctive and disjunctive conditions
1: $L_0 \leftarrow \{\}$; $L_1 \leftarrow CC$
2: $\ell \leftarrow 1$
3: **while** $L_\ell <> \{\}$ **do**
4: **for** condition $\psi \in L_\ell$ **do**
5: compute $R_{\psi 1 \vee 2} \leftarrow R_{\psi 1} \cup R_{\psi 2}$
6: $C_{\psi 1 \vee 2}(\mathcal{L}) \leftarrow FindConversations(R_{\psi 1 \vee 2}, \mathcal{L})$
7: **end for**
8: **if** not interesting according to criteria (B) on $C_{\psi 1 \vee 2}(\mathcal{L})$, *or* $notMon(\psi_{1 \vee 2})$ *or* $TrivUnion(R_{\psi 1}, R_{\psi 2})$ **then**
9: $L_\ell \leftarrow L_\ell - \{\psi\}$
10: **end if**
11: $CC \leftarrow CC \cup L_\ell$
12: **for** conditions $\psi_1, \psi_2 \in L_\ell$ **do**
13: $\psi_{1 \vee 2} \leftarrow \psi_1 \vee \psi_2$
14: **if** $notAssoc(\psi_{1 \vee 2})$ *or* $notInc(R_{\psi 1}, R_{\psi 2})$ **then**
15: $L_{\ell+1} \leftarrow L_{\ell+1} \cup \{\psi_{1 \vee 2}\}$
16: **end if**
17: **end for**
18: $\ell \leftarrow \ell + 1$
19: **end while**

of $\psi_1$ and $\psi_2$ are always less numerous (or of equal number) than those of $\psi_1$ and $\psi_2$ (e.g., consider $\psi_d$ in Figure 6). Hence, it suffices to check if a candidate condition $\psi_{1 \vee 2}$ satisfies $Conv\_ratio(\psi_{1 \vee 2}) > \alpha$ (step 8). In addition, when users provide hints regarding the average length, number or duration of conversations, this information can be used to prune non-interesting conditions.

(2) *Monotonocity of the number and the length of conversations with respect to the disjunctive operator*: The number of conversations resulting from a condition $\psi_{i \vee j}$ is smaller than, and each conversation is at least as long as but expectedly longer than those of $\psi_i$ or $\psi_j$ since disjunctions add to the connectivity of the correlation graph (see, for instance, $\psi_d = \psi_3 \vee \psi_4$ in Figure 6). Therefore, if for a give disjunctive condition the number of conversations does not decrease or the length of (at least some of) conversations does not increase, then such disjunctive condition is not interesting. We use this observation to identify and prune non-interesting disjunctive conditions. This criterion is referred to as $notMon(\psi_{1 \vee 2})$ in line 14 of Algorithm 3.

(3) *Avoiding trivial unions of conversation sets.* Consider node 2 and node 3 in Figure 1, which represent partial views of the process compared to node 5. Applying disjunctive operator on the conditions of these nodes leads to a new node, namely 6 (not shown in Figure 1), which its corresponding conversation set is the union of the conversations in nodes 2 and 3. Such views are mainly interesting for the highest level views (e.g., node 5), where the most complete view of the interaction is desirable. Other intermediate nodes such as node 6 would not add any information and are therefore discarded. Note that users can nonetheless explore unions of any two or more views, if interested, using the exploration tool after completion of the discovery process. Nevertheless, correlation message pairs in node 4 are union of those in nodes 1 and 2 but conversations in 1 and 2 connect together to form bigger conversations in 4. This is because the correlated message pairs in the two sets have common nodes

allowing them to connect and form bigger conversations, e.g., in one set we may have $(m_1, m_2)$ and in another $(m_2, m_3)$. But, in former case, there is no common message in the message pairs of the two sets, and so their conversations do not change. This criterion is referred to as $TrivUnion(R_{\psi 1}, R_{\psi 2})$ in line 8 of Algorithm 3.

**Generating candidate disjunctive conditions**. In this phase, the set of candidate disjunctive conditions for the next level is generated (lines 12 to 17). This set is built by combining selected conditions from the previous level, i.e., the ones that are not pruned. Similar to computation of conjunctive conditions, we avoid computation of disjunction composed of conditions that have been tagged as non-interesting in the previous level, since new disjunctive conditions conditions built using such conditions are necessarily not interesting (producing longer conversations or less numerous than their parent conditions).

We define the following set of criteria to predict which candidate conditions are not potentially interesting, and so to avoid computing the set of conversations for them, which is a computationally expensive operation (see Section 3.3 and Section 5):

(1) *Associativity of conjunction and disjunction*: A condition that combines disjunction and conjunction of the same atomic condition can be simplified into a condition previously explored. For example, the correlation condition $\psi_2 \vee (\psi_2 \wedge \psi_3)$ is equivalent to the correlation condition $\psi_2$ ($R_{\psi_2} \cup (R_{\psi 2} \cap R_{\psi 3}) = R_{\psi 2}$). This is not useful to compute since the conversation sets for above two conditions are the same. Hence, we do not compute such combinations further. This criterion is referred to as $notAssoc(\psi_{1 \vee 2})$ in line 14 of Algorithm 3.

(2) *Inclusion Property*: If $R_{\psi_1}$ is included in $R_{\psi_2}$, i.e., $R_{\psi_1} \subset R_{\psi_2}$, then $\psi_1 \vee \psi_2 = \psi_2$. Therefore, $\psi_1 \vee \psi_2$ is not minimal (since it builds down to $\psi_2$) and we can avoid computing it. Moreover, when $R_{\psi_1} = R_{\psi_2}$ then $\psi_1$ and $\psi_2$ partition the log identically (they produce the same views), i.e., $R_{\psi_1 \vee \psi_2} = R_{\psi_1} = R_{\psi_2}$, and only one is needed to be explored. This criterion is referred to as $notInc(R_{\psi 1}, R_{\psi 2})$ in line 14 of Algorithm 3.

## 3.3 Complexity Analysis

In the following, we analyze the complexity of the proposed correlation conditions discovery approach in both worst case scenarios and also in practical cases.

**Atomic condition discovery**. The time complexity for atomic condition discovery depends on the number of attributes in the dataset and also the size of the log. In the worst case, the time complexity is $O(k^2.p)$, in which $k$ represents the number of attributes in the log, and $p$ is the time complexity of partitioning log $\mathcal{L}$ into conversation for a condition $\psi$. This latter ($p$) consists of computing the set of correlated message pairs $R_\psi$, the correlation graph $G_\psi(\mathcal{L})$, and decomposing it. The worst case time complexity of computing $R_\psi$ is $O(|\mathcal{L}|^2)$, which is the case for comparing each message in the log $\mathcal{L}$ to all messages after it. Building the graph $G_\psi(\mathcal{L})$ from $R_\psi$ has a complexity of $O(|\mathcal{L}|^2)$, as well (adding an edge for all pairs of correlated messages in the graph). Finally, the time complexity of graph decomposition algorithm is $O(|V| + |E|)$, $|V|$ and $|E|$ representing the number of vertices and edges in the graph. This is equivalent to $O(|\mathcal{L}| + |\mathcal{L}|^2)$ in the worst case. Summing up all these, we get the complexity of $O(k^2.|\mathcal{L}|^2)$ for $k$ attributes. At any given time, $R_\psi$ and its corresponding

graph for $\psi$ are in the memory. Therefore, the space complexity, in the worst case, is $O(|\mathcal{L}|^2)$. This is the case for a key-based condition that has only one value, so each message in the log is connected to all other messages.

However, in practice, due to the definition of correlations between messages in information systems in an enterprise, only a small part of all possible combinations makes sense, effective pruning criteria and also efficient implementation techniques, the time and space complexity are significantly smaller that the worst case analysis. The number of attributes with repeating values in the dataset is always much smaller than $k$. If this new number represented by $\kappa$, then we have the number of possible atomic conditions, $\kappa^2/2 \ll k$, and so $k$ can be used as an upper bound for candidate atomic conditions. In addition, not all messages in a log are correlated to each other. Especially, in chain-based correlations, each message is only correlated to one another message in the conversation. Moreover, looking forward from each message in the log, for correlated messages to it, allows to reduce the worst case number of correlated pairs by half, from $|\mathcal{L}|^2$ to $|\mathcal{L}|^2/2$. The combination of these fact, and also using database queries and indexing techniques (e.g., B-Tree in our implementation) to compute $R_\psi$, its time complexity in most practical cases is almost linear with respect to the database size (see Section 5 for experimental results).

**Composite conjunctive condition discovery**. The time complexity of the conjunctive condition discovery algorithm depends on the number of conjunctive conditions, for which the respective set of conversations is computed. Let $s$ be number of such conditions. In the worst case, $s = O(2^{|A|})$, in which $A$ represents the set of atomic conditions. During the computations, $s$ conditions are discovered, so the time complexity for computing the conjunctive conditions is $O(s.p)$, in which $p$ it the time complexity of partitioning log into conversations according to a conjunctive condition $\psi_{1\wedge 2}$. This time complexity is the same of that for atomic conditions (see above). Therefore, in the worst case scenario, the time complexity is $O(2^{|A|}.|\mathcal{L}|^2)$.

However, in practice, given the small number of conjunctive conditions in a system, in an enterprise, compared to huge number of possible ones and also due to pruning criteria, $s$ can be significantly smaller than the worst case analysis shows. In fact, in many systems, not more than few (e.g., 4) atomic conditions are used in forming a conjunctive condition to uniquely represent conversations. Therefore, no more than few levels (4 in this example) have to be explored. In addition, many of the possible candidates are pruned before computing their respective set of conversations, and many others are not even considered as candidates as their parent conditions are pruned.

Similar to computation of atomic condition, there is one set of correlated messages, $R_{\psi 1\wedge 2}$, and its corresponding correlation graph are in the memory at a time. Hence, the space complexity, in the worst case, is $O(|\mathcal{L}|^2)$. Using a similar argument as the one for atomic conditions, the space complexity, in most cases, is significantly smaller than what the worst case scenario shows. In fact, the space complexity is dependent on the size of the longest conversations in a system, denoted by $c_{max}$, which is correlated using a key-based approach. Assuming that all the other conversations are of the same length (the worst case), then approximately there are $|\mathcal{L}|/c_{max}$ conversations in the log. In this case, the required space is $O(\frac{|\mathcal{L}|}{c_{max}}.c_{max}) = O(c_{max}.|\mathcal{L}|)$. In most practical cases, $c_{max} \ll |\mathcal{L}|$.

**Composite disjunctive conditions discovery**. The analysis of worst case and practical case time and space complexities for disjunctive conditions discovery is similar to those for conjunctive conditions discovery. The time complexity, in the worst case scenario, is $O(s.p) = (2^{|C|}.|\mathcal{L}|^2)$, in which $C$ represents the union of the sets of atomic and conjunctive conditions discovered in previous steps. However, in practical cases, $s$ is significantly smaller due to extensive set of pruning criteria applied to only keep the interesting conditions. This number is much smaller than the possible candidates. This is validated by experiments reported in Section 5.

# 4 Visual Exploration and Discovery

In this section, we first present the visualization facilities of the front-end component of the proposed framework. As mentioned before, besides discovering a set of interesting correlation conditions, it is important to help the user in understanding discovered views. We next describe how the tool allows a user to supervise the process view discovery and refine the results. More details on the front-end is provided in Section 5.3.

## 4.1 Process Map

We propose the *process map* as a visualization metaphor for navigating the discovered process views. A process map organizes the various process views based on the relationships that exist between the processes (in our scenario, the business protocols) derived from the conversations corresponding to each view. Figure 1 shows an example of a process map for the `SCM` business service. A process map consists of two visual elements: nodes (process views), and links (relationships between them).

**Process view**. A node represents a discovered process view. Each node is also associated with meta-data that provides a description of its corresponding process view:

- *Statistical meta-data*: These are metrics about the log partitioning into conversations such as the number of conversations, their minimum, average and maximum lengths (i.e., number of messages) and durations (the time difference between the first and the last messages of a conversation);
- *Condition*: The correlation condition that leads to the process view;
- *Correlation approach*: This refers to the approach used for correlating messages, i.e., key-based, reference-based, or a combination of these;
- *Business protocol*: The business protocol of a process view is discovered by applying existing protocol discovery algorithms [25] to the conversations of this view.
- *Service list*: If the WSDL interfaces corresponding to the Web services involved in the messages of the log are known, we can list services which their interaction is represented by the process view. This also allows to identify the dependency between services involved in the given view [28].

**Relationships**. The links between nodes in the map represent the relationships that exist between their corresponding business protocols. These relationships may be of the following type [6]:

- *Subsumption*: Protocol $X$ is subsumed by protocol $Y$ if the set of conversations supported by protocol $X$ is a subset of the conversations supported by protocol $Y$. This relationship allows to specify if one protocol is more specific than another (e.g., nodes 3 and 4 are subsumed by 5 in Figure 1).
- *Part-of*: Protocol $X$ is part-of protocol $Y$ if any given conversation $c$ of $X$ is *part-of* some conversation $c'$ of protocol $Y$. A conversation $c$ of protocol $X$ is part-of conversation $c'$ of protocol $Y$ if messages in $c$ appear in the same order in $c'$. For instance, conversations $\langle m_1, m_2 \rangle$ and $\langle m_2, m_4 \rangle$ are part-of conversation $\langle m_1, m_2, m_3, m_4 \rangle$. Protocols of nodes 1 and 2 in Figure 1 are part-of the protocol of node 4. This relationship highlights that a protocol is a part of a larger (composite) service interaction model (see [7]).

In order to efficiently compute the relationships between protocols of views, we leverage the computation that took place during the condition discovery step. This aspect is very important as, otherwise, we would have to compute protocol relationships between each pair of process views; a very computationally expensive task. Earlier computations can be leveraged since a subsumption of protocols translates into an inclusion property at the conversation set level. During computing composite conditions from $R_{\psi 1}$ and $R_{\psi 2}$, the inclusion between these two is tested (see Section 3.2). In the disjunctive conditions discovery, by default, using the trivial union criterion ensures that the subsumption relationship is only allowed between the process views in the highest level and immediately lower level. Otherwise, between a composite condition $\psi$ and its parent conditions part-of relationship exist in the process map. In fact, conversations corresponding to a correlation condition $\psi_1$ are part-of conversations built using a condition $\psi_{1 \vee 2}$. Conversely, conversations built using a condition $\psi_{1 \wedge 2}$ are part-of conversations built using $\psi_1$. Using this approach, we can conclude on protocol relationships at almost no cost.

**Organization**. We organize the process views by levels such that the highest level contains the largest and most generic protocols (i.e., protocols not subsumed nor are part of any other). Correspondingly, the lower levels represent protocols that correspond to more specific and precise interactions. From any given process view, users can navigate this map by traversing the links from one view to its related views (e.g., when all the discovered conditions are atomic and no relationship exist between them).

To make the job of users easier in identifying what views to visit first, we provide a ranking method which ranks view close to user preferences –in terms of expected average length, number, or duration of conversation–, if provided, higher in the list. This is also helpful when there is no relationship between protocols of discovered views.

## 4.2 User Driven Discovery and Refinement

There are two main reasons for allowing users to supervise the correlation discovery process: (i) users can lead the search toward views that interest them,

and (ii) although the goal in the automated approach is to minimize the risks of false positives (i.e., the inclusion of irrelevant views) and false negatives (i.e., excluding an interesting view), this risk cannot be entirely avoided; there are exceptions to the heuristics discussed in Section 3.1.1. For example, in some applications customerID AND countryCode may be used as correlators. However, countryCode is a categorical attribute with relatively small domain and is therefore likely to be pruned.

We allow users to supervise all steps of the discovery process, from correlator attribute selection, atomic, conjunctive and disjunctive conditions discovery. Before (resp. after) each step, the user can inspect the input (resp. output) and corresponding views, and refine by adding/removing candidate views. This facility proved effective in practice (see Section 5). We propose two refinement operations that allow users to instruct the tool to further explore some directions, after automated discovery is finished, or to remove unrelated results:

- *AddCondition*: Users can propose new conditions to be examined. This operation allows to explore views built based on atomic, conjunctive and disjunctive conditions. In cases, where the proposed condition is examined and pruned during the execution of the algorithm, the reason for which it is pruned is presented to the user to allow for decision making about continuation of exploration.
- *RemoveCondition*. Users can remove a view from the map. It is possible to cascade removal to all children or all parents of the current view.

# 5 Experiments

**Implementation**. We have implemented the proposed process views discovery approach in a prototype system, called `Process Spaceship`. It has been implemented as a set of Eclipse plug-ins using Java 5.0 as the programming language and PostgreSQL 8.2 as the database management system. For the implementation of the graph decomposition algorithm, we used `JGraphT`[1] library (release 0.7.2), which is a free Java graph library that provides mathematical graph-theory objects and algorithms. All experiments have been performed on a notebook machine with 2.3 GHz Duo Core CPU, and 2 GB memory.

## 5.1 Datasets

We carried out experiments on three datasets:

**SCM**. This dataset is the interaction log of a `SCM` business service, developed based on the supply chain management scenario provided by WS-I (the Web Service Interoperability organization)[2], for which a simplified protocol is depicted in Figure 1 (node 5). There are eight Web services realizing this business service. The interaction log of Web services with clients was collected using a real-world commercial logging system for Web services, i.e., HP SOA Manager[3]. The services in `SCM` scenario are implemented in Java and uses Apache Axis as SOAP implementation engine and Apache Tomcat as Web application

[1] `jgrapht.sourceforge.net`
[2] `http://www.ws-i.org`
[3] `http://managementsoftware.hp.com/products/soa`

Table 1: Characteristics of the datasets

| Dataset | SCM | Robostrike | PurchaseNode |
|---|---|---|---|
| service operations | 14 | 32 | 26 |
| messages in log | 4,050 | 40,000 | 34,803 |
| attributes | 28 | 98 | 26 |

server. Table 1 shows the characteristics of this dataset. The log has $4,050$ tuples, each corresponding to an operation invocation. The protocol of SCM has three paths, for which conversations of one path is correlated using disjunctive conditions (the same as those of node 4 in Figure 1), the other using an atomic condition and finally the other is correlated a conjunctive condition (the same as those of node 1 in Figure 1). HP SOA Manager records metadata about message exchange in 13 attributes, and we extracted 15 attributes from messages in this dataset. This dataset provide an example of a system, for which its conversations are correlated in a chain-based method.

**Robostrike**. This is the interaction log of a multi-player on-line game service called *Robostrike*[4]. Clients exchange XML messages with the service. The log contains $40,000$ messages (Table 1). In a pre-processing step, we extracted all the attributes of all messages to present them as a single table. The conversion of XML schema to relational schema is a separate thread of research by itself [21]. In our implementation, we used a simple approach for extracting data elements from XML in which XML structure is ignored and only the name of (deepest) data elements is kept (this simplification was possible in this case since a given element name does not appear in various structures). The XML pre-processing method is explained in Section 5.1.1. This dataset represent a system, which its conversations are correlated using a key-based approach having very long conversations.

**PurchaseNode**. This process log was produced by a workflow management system supporting a purchase order management service called PurchaseNode (PN). The PN dataset contains $34,803$ tuples corresponding to task executions within workflow instances (Table 1). It is private process log of a service in which all messages are correlated using atomic conditions. This 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. By using this dataset we also test the applicability of the approach to process logs. This dataset is an example of a system, for which its conversations are correlated using a key-based approach. It can be considered as typical log of single systems in an enterprise.

### 5.1.1 Pre-processing Step

The main pre-processing step that is performed is processing XML messages in service logs to extract data elements (attributes) and to prepare the relational table that is required as the input of the algorithm. The conversion of XML schema to relational schema is a separate thread of research [21], and we rely on

[4] http://www.robostrike.com

this research for such a conversion in general. In our implementation, however, we used two simplistic but systematic approaches for extracting data elements from XML structure: (i) structure-ignorance naming, (ii) structure-preserving naming. It should be noted that the risk of information loss in using any of the above approaches exists, as XML schemas are richer than relational schemas. This is also the main challenge that XML schema to relational schema conversion approaches is faced.

- *structure-ignorance naming*: In this approach, all the structure of XML is ignored, i.e., XML schema of each messages is flattened and only data elements (including element attributes) are extracted and considered as an attribute in table $\mathcal{L}$. This approach is appropriate for applications that a same element/attribute name is used with the same meaning throughout the whole schema. For instance, the element customerID refers to the same data element, which is the customer identifier in any place of the schema. In this case, the corresponding name of this element is also customerID in table $\mathcal{L}$.

- *structure-preserving naming*: In this approach, the name of XML constructs is also considered as part of the attribute name. For example, if there is a complex XML construct called Order and there is an element of this construct called, customerID, then the corresponding name of attribute in $\mathcal{L}$ is considered Order_customerID, which will be treated differently from another attribute called Invoice_customerID. This method of naming attributes is appropriate for XML schemas that do not use the same element/attribute name for the same real-world entity or in the cases that the naming is not complete. For example, both Order and Invoice construct may have an element called Number. Then, it is desired to distinguish between Order_Number and Invoice_Number. In our implementation, it is possible to specify how many levels of constructs to be considered in the naming as the input of the procedure.

In reality, a combination of these approaches may be required as there is some semantics are involved in naming of attributes that may not be able to address properly. In our experiments, in `Robostrike` and also in `SCM` we have used the structure-ignorance naming method, as it was an appropriate selection considering the semantics of naming attributes. However, our approach allows for manual editing the XML conversion as the input of the algorithm is table $\mathcal{L}$, and in general we assume that pre-processing is performed correctly.

## 5.2 Evaluation of the Discovery Process

**Evaluation criteria**. We evaluate our proposition along three dimensions: (i) the quality of discovered views, (ii) the execution time, and (iii) the contribution of the proposed criteria in pruning the search space.

The quality of the results is assessed using classical *precision* and *recall* metrics. Precision is defined as the percentage of discovered views which are actually interesting. Recall is computed as the percentage of interesting views that have actually been discovered. We manually selected the set of views that could be tagged as interesting, considering a wide range of user considerations, i.e., covering all the meaningful ways by which messages of the log could be correlated

into processes. This was performed in collaboration with dataset owners. We ran experiments to validate the performance of the proposed approach on the above datasets. The following highlight the main findings of these experiments.

**The quality of discovered views**. For dataset `PN`, by applying the approach for correlator attribute identification (based on criterion ($A$), there are 4 attributes out of 26 selected. Then, one of these attributes ($nodeInstanceID$, which is the message unique identifier) also pruned based on having all unique values in the dataset. The three remained attributes are $flowInstanceId$, $startTime$ and $endTime$ attributes. The reason that $startTime$ and $endTime$ attributes are selected is that, in this dataset, there are some tasks that were started or finished at the same times, so these two attributes have repeated values. There are four conditions defined based on these three attributes, namely $flowInstanceId = flowInstanceId$, $startTime = startTime$, $endTime = endTime$, and $startTime = endTime$. However, by applying criterion ($B$), on the number of conversations formed based using each conditions, only condition $flowInstanceId = flowInstanceId$ remains. In fact, most of conversations built using other conditions are of length 1 (messages are not correlated). This results shows a recall of 100% and precision of 100%. In fact, the only discovered condition is the only interesting correlation to discover for this dataset.

For `Robostrike` dataset, the recall is 90% (9 out of 10 were discovered). One of the expected atomic conditions was not discovered. The reason is that it corresponds to a categorical attribute that was pruned due to small number of distinct values. There are 7 key-based atomic conditions discovered. These correspond to views representing conversations in individual games, individual user sessions (that include several games), and multiple sessions of a same player. There are three conjunctive conditions discovered, each consisting of two atomic conditions, and one of them is interesting. It corresponds to a view that identifies individual games of each user allows to analyse the behavior of a given user in a game. There are two disjunctive conditions discovered, each consisting of two atomic conditions. One of them is an unexpected view that was surprising for the dataset owners. It corresponds to a correlation of messages based on private chat conversations among players. This view highlights the communities of players that are talking about a given game. In total, for this dataset, the precision was 75% (9 out of 12 were interesting).

For the `SCM` dataset, 9 atomic conditions are selected (after applying criterion ($A$)). The attributes are $customerID$, $quoteID$, $oID$, $invID$, $payID$, $shipID$, $custID$, $surveyID$, and $rfpID$. These are used to form 9 key-based atomic conditions. The atomic condition based on $customerID$ is used to correlate messages in the catalogue system. Atomic conditions based on $quoteID$, $oID$, $invID$, $payID$, $shipID$ are used to correlate messages in the quoting, ordering, invoice, payment, and shipment systems. There is one conjunctive condition discovered that is formed from conjunction of $customerID$ and $surveyID$. This condition correlate messages in the customer relationship management (CRM) system. In fact, $customerID$ is used in two systems for correlations, i.e., the catalogue and in the CRM systems. Although, in the latter system it is used as part of a conjunctive condition.

Finally, there are 11 disjunctive conditions discovered. One of these conditions is the disjunction of atomic conditions based on $customerID$, $quoteID$, $oID$, $invID$, $payID$, and $shipID$. This condition corresponds to the *Retailer* business service offering all purchase order management services in the enter-

prise (see Figure 1). Another disjunctive condition, formed based on atomic conditions on $custID$ and $rfpID$, is related to the view corresponding to the product system. The view with the highest number of conditions is formed based on disjunction of all above conditions. This view represent the model of interactions in the whole `SCM` scenario. There are 9 disjunctive conditions that represent intermediate views in the process map. For instance, they represent the interactions between the quoting and ordering systems, the ordering and invoice, invoice and payment, and so on. These results shows a recall of 100% as all paths of the `SCM` protocol are characterized correctly. Evaluating the precision depends on the goal of analysis. It can be seen as 100% as no non-interesting views, in general, were discovered. However, if 9 intermediate views in the map are considered as non-interesting, then the precision is 42%. In fact, the main source for having to a low precision is the correlator attribute identification step. It is possible in many cases that some attributes qualify the statistical criteria, but they are not relevant for correlation, or some relevant attribute are pruned since they do not qualify the criterion ($A$). Examples of non-relevant attributes that qualify the statistical criteria, in `SCM`, dataset are $requestSize$, $responseSize$ (from metadata attributes provided by HP SOA Manager). If such attributes are pruned manually, in this step, a great amount of computations is saved in the later stages. If this is done, the approach proposed for computing composite conditions achieves a very high precision. The good news is that identifying non-relevant attributes is easy for a user and it is facilitated by various meta-data information provided by the tool, see Section 5.3. Based on this, in above experiments, we have omitted above mentioned two attributes in this step.

**Execution time**. Figure 7 shows the execution time of the approach on the `PurchaseNode` dataset for 6 different size of the database. In this case, there is only one atomic condition discovered. Hence, only the execution time of atomic conditions discovery is presented. This chart allows to compare the amount of time spent on different sub-steps, i.e., computing $R_\psi$, $G_\psi$, $C_\psi$, and applying pruning criteria (denoted by `Other`). It can be seen that more than 90% of the time is spent on computing $R_\psi$. This step is performed as an SQL query over the database `PurchaseNode`, and the other steps are mainly performed in the memory. $R_\psi$ for condition $flowInstanceID = flowInstanceID$ takes the longest time, among others, due to longer length of conversations. $R_\psi$ for this condition has $48,476$ rows on the dataset with 15000 messages. The selected attributes and discovered conditions are the same for all dataset sizes. The time denoted by `Other` represent the time spent for applying pruning conditions (here, criterion ($A$) and criterion ($B$)). This time is at the same range of that of computing $G_\psi$, but less than those of computing $C_\psi$ and $R_\psi$. In general, the time increase is nearly linear as the dataset size increases.

The overall execution time of the approach on the `Robostrike` dataset is shown in Figure 8. The size of the dataset is varied from 2500 messages in the log to 15000 messages. This chart also compares the execution time for the three sptes, i.e., atomic, conjunctive, and disjunctive conditions discovery. In average, 36% of time is spent on discovering atomic conditions, 20% on conjunctive conditions, and 44% on disjunctive conditions. Figure 9 shows the details of execution time of atomic conditions in this dataset. In average, the shares of computing $R_\psi$, $G_\psi$, $C_\psi$, and applying pruning criteria (`Other`) are 81%, 14%, 1.5% and 3.5% of the time, respectively. The reason that computing

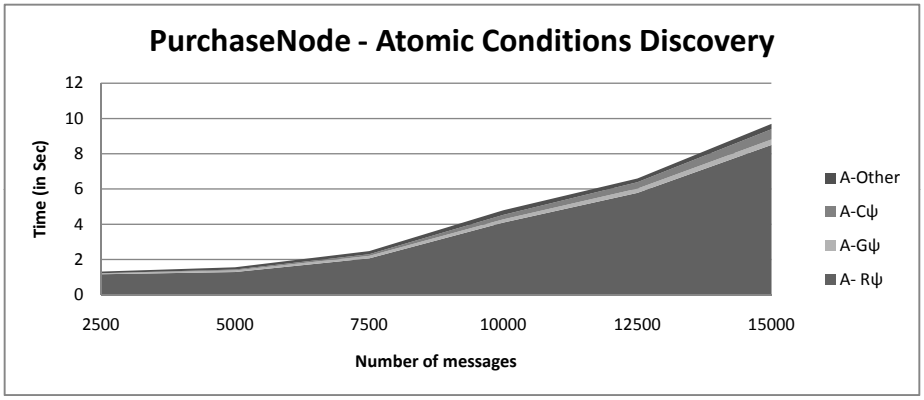



Figure 7: The execution time of the approach on the `PurchaseNode` dataset

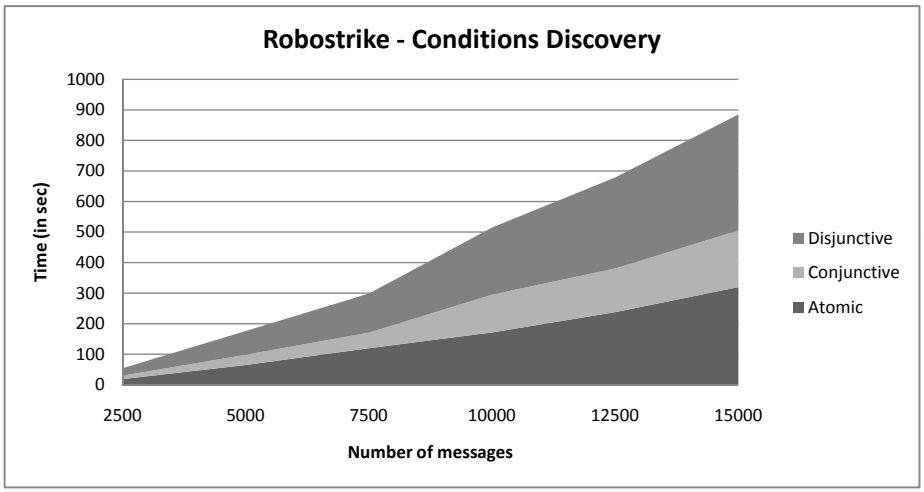



Figure 8: The execution time of the approach on the `Robostrike` dataset - all steps

$R_\psi$ takes this amount of time is that atomic conditions in *Robostrike* are key-based, and there are very long conversations in this dataset. For instance, the maximum length of conversations for an atomic condition in the dataset with 15000 messages is 821 messages, and $R_\psi$ of this condition has $2,140,502$ rows. Having this long conversation is normal in this application, as usually during a game many messages are communicated between the service and the application of the player.

Figure 10 shows the details of execution time of the conjunctive conditions discovery on this dataset for varying sizes. In this phase, in average, 48% of time is spent on computing $R_\psi$, 11% on computing $G_\psi$, 1% on computing $C_\psi$ and 40% on applying pruning criteria. The reason that applying pruning criteria has a significant share in the execution time is that applying some of criteria, e.g., checking for inclusion, require executing queries on the database. Note that the correlation graph in this case is less connected and so the time for computing

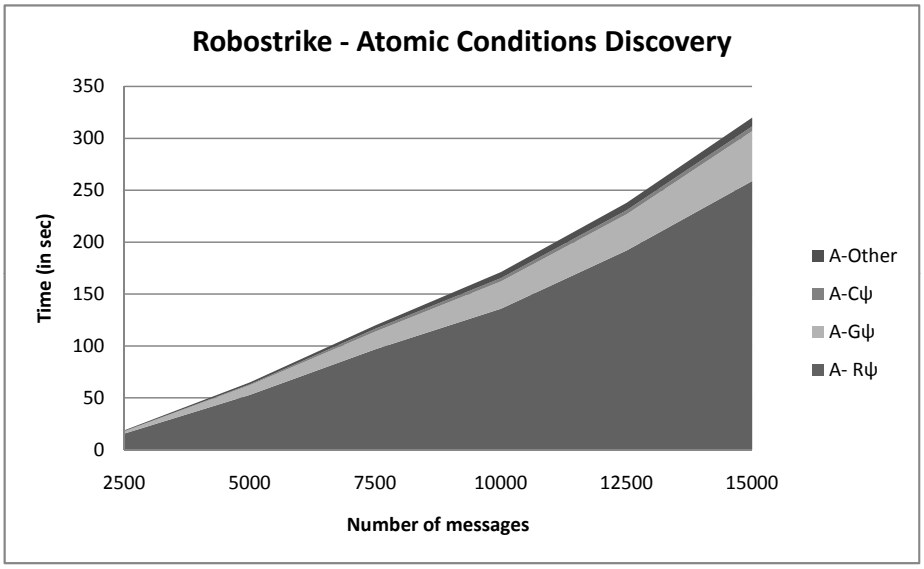



Figure 9: The execution time of the approach on the `Robostrike` dataset - atomic conditions discovery

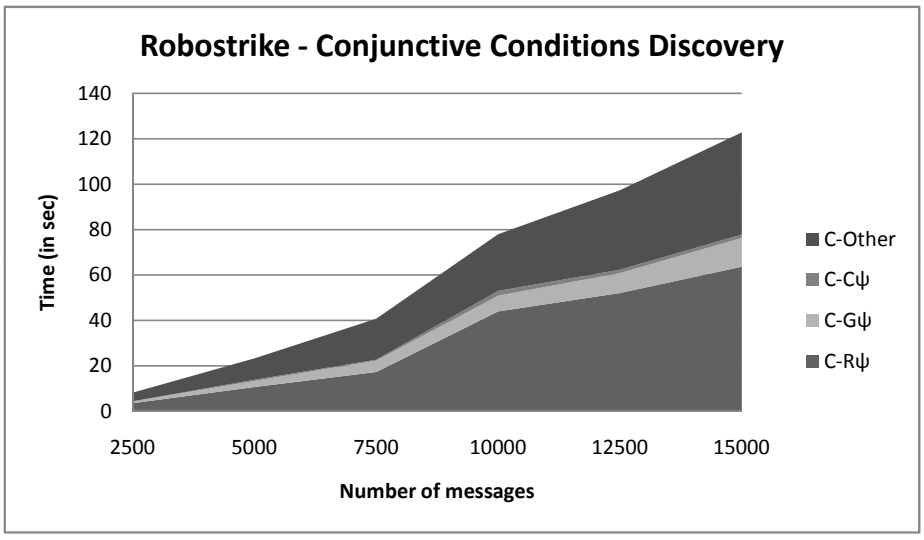



Figure 10: The execution time of the approach on the `Robostrike` dataset - conjunctive conditions discovery

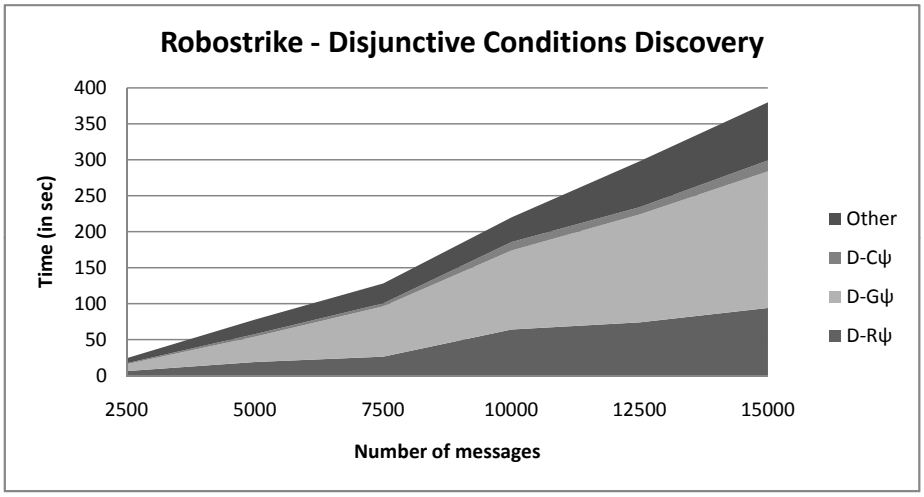



Figure 11: The execution time of the approach on the `Robostrike` dataset - disjunctive conditions discovery

connected components is small. Finally, Figure 11 shows the details of execution time of the disjunctive conditions discovery on this dataset for various number of messages. This chart shows that 25%, 48%, 4% and 23% of the time, in average, are spent on computing $R_\psi$, $G_\psi$, $C_\psi$, and applying pruning criteria, respectively. It can be seen that most of the time is spent on computing the correlation graph (the graph is built by parsing $R_\psi$ and creating an edge for each correlated message pairs in $R_\psi$). The reason for this significant time is that the correlation graphs for disjunctive conditions can be significantly big (since the set of edges of a correlated graph for a disjunctive condition is the union of those of its parent conditions) in this dataset, given the key-based correlation and long conversations.

As explained above, significant time of the algorithm is spent on computing $R_\psi$ in all steps, and also on $G_\psi$ in computing disjunctive conditions. One way to reduce this time is to use approximate approaches for computing $R_\psi$ and also $G_\psi$. Investigation of these optimization techniques is proposed as part of future work (see Section 7).

Experiments shows that the discovered conditions are the same for the datasets with 7500 messages and higher. The reason is that in the `Robostrike` dataset, playing session of users typically lasts between 1 and 4 hours, that is, about $7,000$ messages in the dataset. Hence, if we have $7,000$ or more messages, then we have complete conversations in the log. Identifying the appropriate size of a dataset, in which there exist complete conversations, is a domain specific task. Our tool allows specifying either the size of the dataset to use in terms of number of messages, or in terms of the duration (the time difference between the first message and the last message in the selected sample of the dataset).

Table 2 shows the execution time of the approach on the `SCM` dataset. This dataset represent a case, where conversations are correlated using a chain-based approach. Hence, in this case, the execution time of the atomic conditions discovery is small (0.859 Sec.), compared to 63 Sec. for a sample with the same

Table 2: The execution time of the approach on the SCM dataset

| | $R_\psi$ | $G_\psi$ | $C_\psi$ | Other | Total |
|---|---|---|---|---|---|
| Atomic | 0.562 | 0.016 | 0.031 | 0.25 | 0.859 |
| Conjunctive | 0.079 | 0.002 | 0.001 | 0.34 | 0.422 |
| Disjunctive | 0.606 | 0.422 | 0.205 | 1.22 | 2.453 |

size from Robostrike dataset. For this dataset, 66% of the time is spent on computing disjunctive conditions, 11% on computing conjunctive conditions, and 23% on computing atomic conditions.

Above shows that the proposed approach is efficient, especially, in cases where the correlation follows a chain-based approach. The reason is that in such cases atomic conditions have conversations of relatively small lengths. In addition, the proposed approach is very in pruning the search space for computing composite conditions. Hence, in steps for composite condition discovery, $R_\psi$, $G_\psi$ and $C_\psi$ are only computed for relevant conditions.

**Search space pruning**. As discussed above, the proposed criteria proved effective in identifying and pruning non-interesting candidates. We analyzed the contribution of individual criteria to the pruning for each dataset. For the PurchaseNode dataset, only criterion ($A$) and criterion ($B$) are applied for atomic conditions. Here, we only evaluate the criteria used for pruning the search space for computing composite conditions. Figure 12 shows the usage of proposed criteria in pruning the search space in discovery of conjunctive conditions for Robostrike dataset. In this case, criterion ($B$) is responsible for 31%, inclusion for 25%, monotonocity for 25%, and attribute definition constraints for 18% of pruning. These numbers show that almost all criteria are equally important for search space pruning in discovering conjunctive conditions for this dataset. In discovery of disjunctive conditions, criterion (B) has a share of 4%, inclusion 13%, monotonocity 50%, associativity 9% and finally trivial union criterion 23% (see Figure 13). This shows that monotonocity and trivial union criteria are most useful criteria for pruning the search space in discovering disjunctive conditions for systems that messages are mainly correlated using key-based conditions.

For SCM dataset, the contribution of pruning criteria in discovering conjunctive conditions are as follows (see Figure 14): attribute definitions constraints 80%, monotonocity 3%, inclusion 0, and criterion ($B$) 17%. The fact that attribute definition constraints palyed a significant role is consistent with the observation that the conjunction of many of attribute cannot be defined as they are defined for messages of different systems, e.g., for one quoting and another for ordering system. Inclusion played no role, in this case, as the set of correlated messages for each condition do not have overlap, again because they are related to different systems. Figure 15 shows the contributions of pruning criteria in discovery of disjunctive conditions for this dataset. This shows that trivial union accounts for 88% of pruning, associativity 9%, monotonocity 0, finally criterion ($B$) 3%. The significant use of trivial union also stems from the fact that the set of conversations in vast majarity of possible disjunction of conditions is simply the union of the set of conversations of their parent conditions. This is because such parent conditions correspond to un-related systems, so no longer conversations can be formed. Note that the only view that is excluded

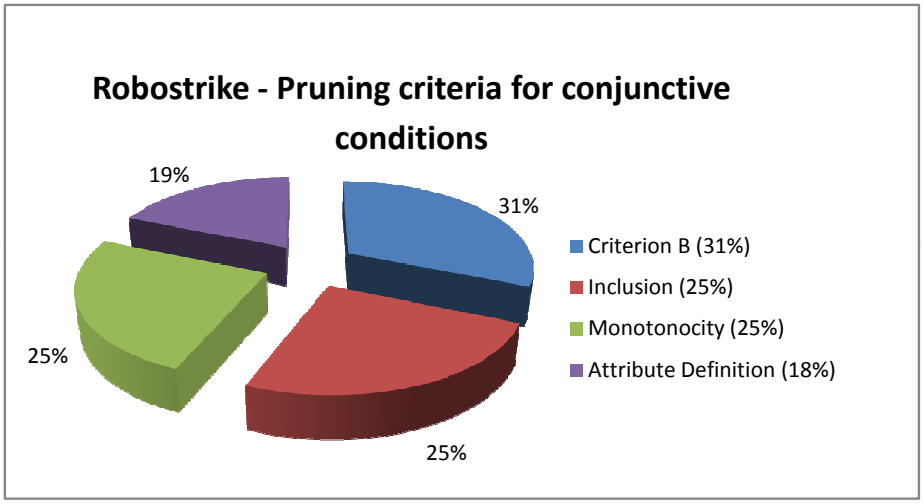



Figure 12: The contributions of criteria in pruning the search space of conjunctive conditions - `Robostrike` dataset

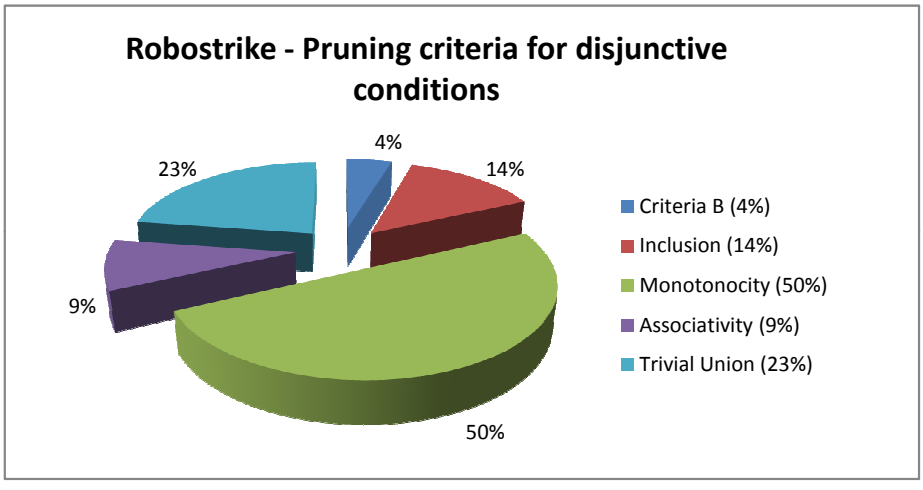



Figure 13: The contributions of criteria in pruning the search space of disjunctive conditions - `Robostrike` dataset

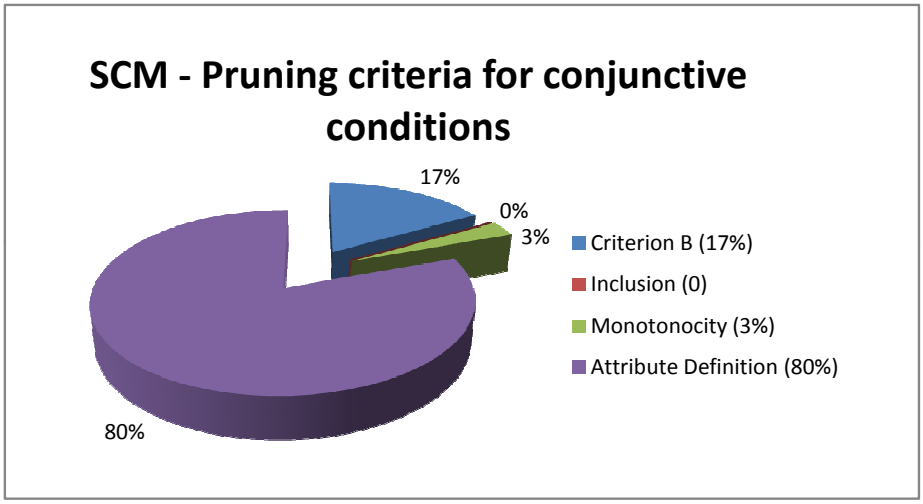



Figure 14: The contributions of criteria in pruning the search space of conjunctive conditions - SCM dataset

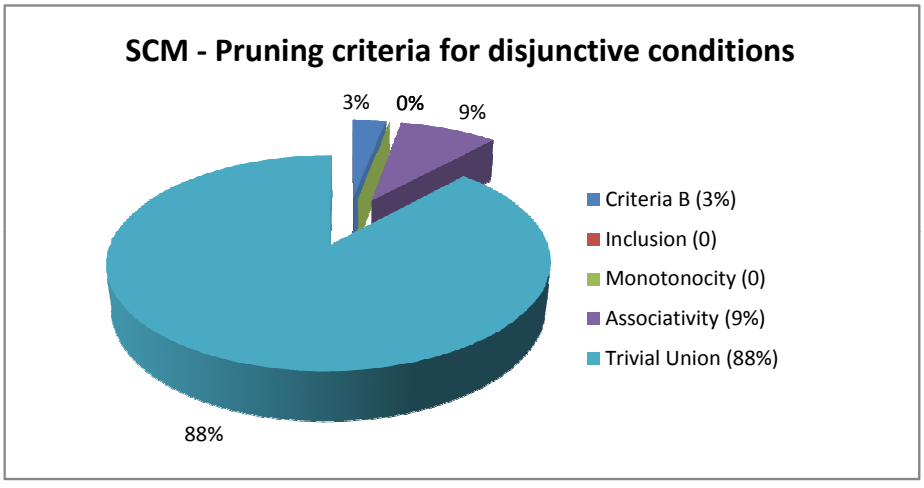



Figure 15: The contributions of criteria in pruning the search space of disjunctive conditions - SCM dataset

from the trivial union criterion, in the automated disjunctive conditions discovery approach, is the most top view of the process map. This view corresponds to the view of interactions across the whole systems (see Section 3.2.3).

## 5.3 Experience using `Process Spaceship`

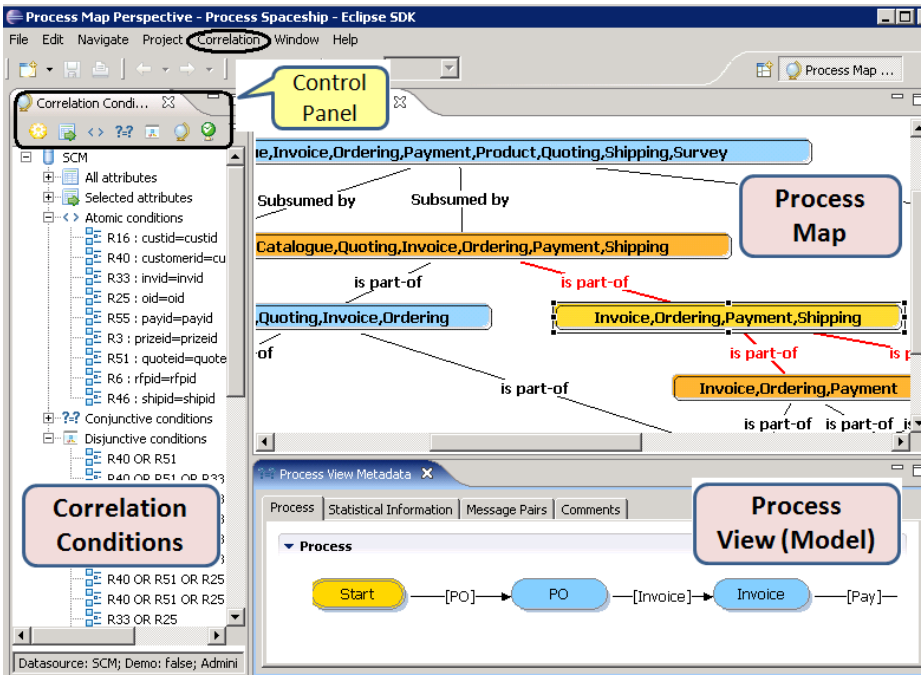
The `Process Spaceship` implements the proposed approach, and provides visual facilities to discover and refine process views. The tool can be used by a process space administrator. This role can be taken by a *process architect*, who is an actor responsible for design and management of process models in the enterprise. In the following, we show how `Process Spaceship` simplifies the job of the process architect. As a demonstration showcase, we use the `SCM` dataset.

The process architect starts from the integrated event log and can operate the tool in two modes: *automated*, or *semi-automated*. In the automated mode, the architect can instruct the tool (using buttons in the top-left in Figure 16(a), which shows a screenshot of the front-end of the tool) to automatically discover the set of potentially interesting process views using the heuristics. The discovered process views are organized in a process map, similar to the one in Figure 1.

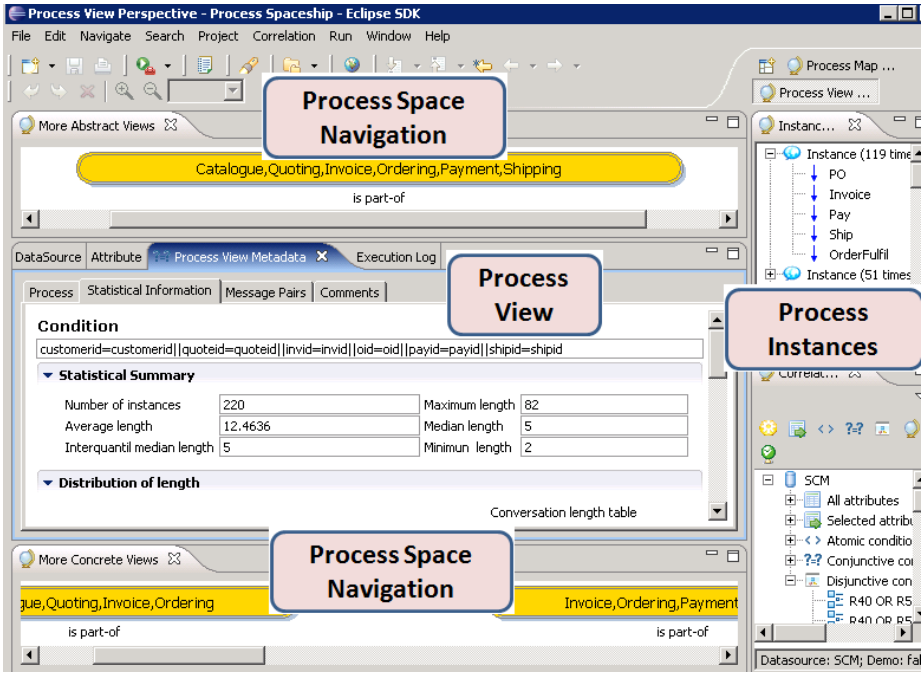
The job of the architect is to explore the map, and to refine it by identifying interesting ones, removing un-related or non-interesting views (e.g., some middle-level views that correspond to partial views of some process), or augmenting process views with comments that might be useful for end-users. In the case of `SCM`, the architect may start from one of the bottom level views, or the highest view, which corresponds to the process of the whole enterprise. Then, using the links in map, she can navigate through process views. Upon selection of a view in the map, related process views are highlighted. The zoom-in and zoom-out operations in the map are provided by following the links, from a given process view, downwards or upwards to access more concrete or more abstract views, respectively.

For `SCM`, 7 out of 9 bottom-level views correspond to views of individual (sub)systems, which are interesting to keep in the map. These can be quickly identified by looking at the map, as nodes in the map are labeled with the names of systems that they are representing. In addition, when a view is selected, the meta-data related to it (e.g. its process model and statistical meta-data) are displayed in the bottom-right frame. Alternatively, the architect can switch to a process view centric view, shown in Figure 16(b), where all meta-data are presented in a single environment. In this view, the top and bottom frames show immediately related views (immediately higher and lower views in the map), and their relationships with the current view. These allow to navigate the map, while focusing on understanding a given view. To provide more insight on a process view, top-k frequent process instances of a view are also available (the top-right frame in Figure 16(b)).

After visiting the bottom level views, the architect may look at the highest level view, which corresponds to the `SCM` view. The architect may decide to hide the some middle level nodes, which correspond to partial views of `SCM` process, e.g., showing only interactions of payment and shipping subsystems, probably because these may not be frequently used, or to remove them if never needed for analysis, so to keep only 10 views visible in the map. The architect can propose new process views for consideration by the tool, if there are views that




(a) Process Map Centric Screenshot




(b) Process View Centric Screenshot

Figure 16: Screenshots of `Process Spaceship`: the Process Space Discovery System

are not discovered. This can be done by selecting the name of a system, desired messages or attributes that form the corresponding correlation condition.

In the semi-automated mode, the architect can supervise the process view discovery. This is conducted in three steps including candidate attributes selection, simple conditions and composite conditions discovery. We show that the tool can capture the architect's knowledge in terms of expected correlation pattern (key-based or reference-based), the average number, duration or length of process instances for various systems. This information is used to direct the search towards desired process views. In addition, before and after each step, the architect is provided with a set of meta-data (including statistical meta-data, process models, and coloring schemes) that helps in making informed decisions, e.g., to keep a condition for consideration or to remove it. This interactive discovery and refinement allows to effectively discover interesting process views and to avoid discovery of un-related views. Ranking of process views, according to the user's interests, also helps in effectively exploring the process maps.

Above facilities save considerable amount of time and efforts when compared to what should be done without such tool support. Integrating this tool with available process analysis and tracking tools (e.g., [18, 4, 5, 17]) also simplifies the job of end-users since using the map and the process view centric view make it easy to locate the desired views for subsequent analysis.

# 6 Related Work

To the best of our knowledge, this is the first work to propose an approach for analyzing the logs of web service interactions by automatically discovering and organizing models of these interactions at various levels of abstraction. The closely related work are discussed in the following:

**Functional dependencies**. The problem of correlation dependency analysis is related to that of discovering functional dependency. In functional dependency inference [20] 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. Approximate functional dependency relax this constraints and look for properties that hold for "most" of the tuples. Many types of functional dependencies (e.g., nested, multivalued, join, etc.) have been explored and algorithms have been proposed to infer them. Discovering a functional dependency is interesting as it reveals an implicit constraint over attribute values of tuples, which are not expressed in the schema. In contrast to dependency, 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. In fact, there may exist valid functional dependencies between some messages, but these dependencies do not reveal anything about the conversations that these messages form. In fact, values of some attributes of messages may show valid dependency, but these messages may not be part of a same conversation.

**Composite key discovery**. The problem of discovering (composite) keys in relational databases [30] consists of identifying groups of attributes such that, taken together, they identify individual tuples. While related, this problem has important differences with respect to that of message correlation: whether a group of attributes is or not a valid composite key can be assessed objectively (it has to identify a unique tuple). By contrast, there are only statistical criteria

to decide on the validity of a process view and to be efficient, an exploration of the solution space can only be achieved by leveraging properties (such as inclusion or monotonocity) specific to the problem of message correlation. In addition, key-based correlation of messages into conversations is only one of the possible types. Correlating messages using a chain-based approach into conversations is another example (see Section 2.3.2).

**Association rule mining**. Association rule mining techniques identify values that co-occur frequently in tuples of a dataset [16]. However, frequent co-occurrence of values is not a sufficient criteria to identify all the correct process views. For instance, in the reference-based correlation method a value used for correlating two messages appears only twice in the entire dataset (once for each of the two messages). In addition, while association rule mining looks for values that co-occur in a same tuple, message correlation is concerned with values that occur in different messages in the whole database.

**Classification**. Since conversations are drawn from partitioning the log, building them could be seen as a classification problem [1] where each conversation is a distinct class. 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, messages are sometimes correlated by reference to each other rather than by reference to the class (conversation), making it impossible to define a classification function on a message-by-message basis. 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 conditions from a collection of conversation instances, if available, would be an interesting and complementary problem to explore.

**Clustering**. One might also argue that correlation could be formulated as a clustering problem [19]. In clustering, the relationship between members of a cluster is typically assessed by their relative proximity according to, e.g., 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 conversations may be very similar (e.g. two purchase orders for the same products). In fact, two messages of a same conversation may well have nothing in common due to the transitive nature of the correlation mechanism. Hence, clustering approaches—as well as other similarity-based approaches such as, e.g., record linkage [12]—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, which is the purpose of this paper. We have performed some experiments trying to address the correlation problem as a clustering problem that supports this claim. The result of experiments is presented in Appendix A.

**Session reconstruction**. Web usage mining has raised the problem of *session reconstruction* [31]. 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 information is present in the log, but it is buried among other irrelevant attributes and the problem is that of discovering which attributes or combination thereof are the useful ones.

In time-based session reconstruction, the main issue is 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 [11], the problem of service session reconstruction based on time-based approach is investigated. It is mentioned that unlike human sessions in the Web, for which it is possible to specify some session boundaries, e.g., maximum duration, it is not possible to take a similar approach for Web services. In Web services the sessions range from very short to very long. The proposed approach in this work to heuristically specify a session duration threshold, and then evaluate the quality of the sessions based on the assumption that the result sessions should be similar in terms of session duration, number of services consumed in a session, the order of service consumed and the like. The threshold is then updated until sessions with a satisfactory quality are found. One limitation of this approach is that it only allows to discover sessions in cases, where all the user sessions are well separated, e.g., there is no concurrent sessions with a same service, and also when sessions are similar. These covers a small portion of all possible cases. In addition, no experimental result of this approach is reported. We argue that the time information by itself is not enough and the message content also should be considered, as proposed in this paper. In fact, these two types of information are complementary (see Section 7).

**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 [22] or event correlation for root cause analysis [32]. In this paper, correlation means grouping messages that belong to the same conversation. It is possible that events that are identified as dependent in above approaches are not part of the same conversations, or events related to a same conversation are not recognized as dependent as they may occur very far from each other.

**Message correlation in Web services**. The need for automated approaches to message correlation in Web services has first been reported in [24] where a real situation on how to correlate service messages is presented. Correlation patterns in Web service workflows is studied in [3]. In this work, various correlation mechanisms in Web services workflow are characterized in three categories of function-based, chain-based and aggregation functions. This categorization covers the correlation mechanisms discussed in our approach, however, no automated support for message correlation is proposed.

The need for automated approaches for correlation of service messages in composite business applications is also raised in IBM Websphere platform [10]. This work also proposes an initial approach for discovery of correlation identifiers in messages from the log of service interactions. In this approach the content of each message type (e.g., *PurchaseOrder*, or *Invoice*) is extracted from messages of this type and stored in a table. Then attributes in one table are considered with attributes in other tables (message types) in order to discover correlations between them. This is akin to discovery of atomic correlation conditions in our approach. The end result of this approach is identifying the correlation between message types (e.g., *PurchaseOrder* and *Invoice* message types). In addition to the fact that our approach for messages correlation in

Web services (found in [26]) is reported before this approach in the literature, our approach has several important advantages compared to this approach: (i) this approach only considers atomic conditions (corresponding to the first step in our approach), while our approach also considers composite (conjunctive and disjunctive) conditions; (ii) it identifies correlations between pairs of message types, while we reason at the conversation level and use many properties of conversation to identify interestingness of a correlation condition; (iii) we introduce the notion of process views to cater for the fact that there are more than one possible way of correlating messages into conversations, and hence discover different correlation conditions to form conversations. A process view allows users to find out the views that are interesting for their analysis, however, using this approach it is not clear how messages are correlated at a conversation and process level.

# 7 Conclusions and Future Work

In this paper, we have presented concepts, a framework, algorithms, as well as a system, called `Process Spaceship`, for discovery of a process space, focusing on the analysis of interactions among services captured in their interaction logs. We have characterized the problem in terms of identifying process views that are views over the process spaces, expressed in terms of different ways to group messages into service conversations. The main contributions and results, besides the framing of the problem, lie in (i) the identification of correlation conditions for service execution data, (ii) the presentation of an efficient and viable approach for the discovery of process views by adopting a level-wise approach, and (iii) the organization of process views into a process map, labeled and structured in terms of the conversations and models implied by the correlations associated to each node in the map. Based on this framework and the `Process Spaceship` implementation, we can look at a set of logs, identify process views, and users can choose the ones that are appropriate to the kind of analysis that is desired, at different levels of abstraction.

Research in the area of process spaces is just starting, and this paper is a first step in the direction of realizing a *process space management system* for Web applications, to allow the analysis of interactions and business process execution data in an enterprise. Further steps needed in this direction involve providing a query mechanisms and system for performing OLAP process analysis over heterogeneous and distributed data (relational data, XML, MS Word documents, and so on), and a monitoring system that allows real-time tracking of executions at different abstraction levels based on the correlation perspectives. In particular, the process space discovery framework can be extended in many directions. These extensions make it possible to adapt it in various contexts. We highlight below some future research directions:

*Condition 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 data. The condition language used in this article is one of the 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 conditions would replace the current graph structure with a weighted graph and the

problem of identifying connected components is replaced by that of identifying strongly connected components, i.e. components such that the aggregate of their relationship weights is above some threshold.

*Time constraints on correlation conditions.* Discovering condition 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 messages in order to create log partitions. This way, messages 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 messages of each partition have a time difference larger than the average time difference of the messages across all partitions of the log.

*Extension to heterogeneous data.* We performed the analysis on service logs, but the same concepts with variations can be applied to generic process execution data from heterogeneous data sources such as emails, Word documents, text documents, etc. The next step in our research consists therefore in identifying a methodology and in developing tools for pre-processing enterprise data so that it can be fed to the algorithms discussed here.

*Optimization and approximation techniques.* The approach for computing the set of correlated message pairs ($R_\psi$) is to include all pairs of correlated messages in $R_\psi$. For long conversations that are correlated using a key-based approach, $R_\psi$ becomes large (if $c_{max}$ is the size of the longest conversation, then the size of correlated message pair only for this conversation is $(c_{max})^2$). However, in some cases, it is possible to devise some optimization techniques to do not include all the correlated message pair to compute $G_\psi(\mathcal{L})$. For instance, assume that we have $R_\psi = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4), (5,6)\}$. In this case, $C_\psi(\mathcal{L}) = \{\langle 1,2,3,4\rangle, \langle 5,6\rangle\}$. In fact, to compute $C_\psi(\mathcal{L})$ it is sufficient to have $R_\psi = \{(1,2), (1,3), (1,4), (5,6)\}$, and not including pairs $(2,3)$, $(2,4)$, $(3,4)$ does not change the results due to transitivity relationships between message pairs. In some other cases it may be appropriate to use approximate approaches instead of exact approaches proposed in this paper. This may include using a smaller sample of dataset to estimate the number of conversations, but not to actually compute them. Devising and applying such techniques may reduce both time and space complexity of computing graph $G_\psi(\mathcal{L})$.

**Acknowledgement 1** *The authors would like to thank Pierre Buraud and Angela Chen for their help in implementation of the approach presented in this paper.*

# References

[1] R. Agrawal and et al. An interval classifier for database mining applications. In *VLDB*, 1992.

[2] P. Andritsos, R. J. Miller, and P. Tsaparas. Information-theoretic tools for mining database structure from large data sets. In *SIGMOD Conference*, pages 731–742, 2004.

[3] A. Barros and et al. Correlation patterns in service-oriented architectures. In *FASE*, 2007.

[4] C. Beeri and at al. Querying business processes. In *VLDB*, 2006.

[5] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Query-based monitoring of BPEL business processes. In *SIGMOD Conference*, 2007.

[6] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *Data Knowl. Eng.*, 58(3), 2006.

[7] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB*, pages 613–624, 2005.

[8] D. Beyer and et al. Web service interfaces. In *WWW*, pages 148–159, 2005.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[10] R. H. Y. De Pauw, Wim; Hoch. Discovering conversations in web services using semantic correlation analysis. In *IEEE International Conference on Web Services (ICWS)*, pages 639–646, July 2007.

[11] S. Dustdar and R. Gombotz. Discovering web service workflows using web services interaction mining. *IJBPIM*, 1(4):256–266, 2006.

[12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[13] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

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

[15] A. Y. Halevy and et al. Enterprise information integration: successes, challenges and controversies. In *SIGMOD Conference*, 2005.

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

[17] HP. HP OpenView solutions. In *www.managementsoftware.hp.com*.

[18] IBM. FileNet enterprise content management solutions. In *www.filenet.com*.

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

[20] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. In *ICDT'92*.

[21] D. Lee, M. Mani, and W. W. Chu. Schema conversion methods between xml and relational models. In *Knowledge Transformation for the Semantic Web*, pages 1–17. 2003.

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

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

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

[25] H. Motahari, R. Saint-Paul, B. Benatallah, and F. Casati. Protocol discovery from web service interaction logs. In *ICDE 07*.

[26] H. Motahari, R. Saint-Paul, B. Benatallah, F. Casati, and P. Andritsos. Message correlation for conversation reconstruction in service interaction logs. Technical Report UNSW-CSE-TR-0709, The University of New South Wales, March 2007.

[27] Oracle. Oracle BPEL Process Manager. In *www.oracle.com/technology/bpel*.

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

[29] S. Sahar. Interestingness via what is not interesting. In *KDD '99*, pages 332–336, 1999.

[30] Y. Sismanis and et al. Gordian: Efficient and scalable discovery of composite keys. In *VLDB*, pages 691–702, 2006.

[31] M. Spiliopoulou and et al. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *INFORMS J. on Computing*, 15(2):171–190, 2003.

[32] M. Steinle and et al. Mapping moving landscapes by mining mountains of logs: Novel techniques for dependency model generation. In *VLDB*, pages 1093–1102, 2006.

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

# A Correlation Problem as a Clustering Problem

In this section, we present some of our results when we tackle the correlation problem as a clustering problem. In performing clustering according to the traditional way, our goal is to group together conversation messages, which are deemed very similar and separate the dissimilar ones in different clusters. This is sometimes called *horizontal* clustering. Driven by a similarity measure the clustering algorithm places together those messages that reveal maximum similarity.

In our problem, we performed horizontal clustering on the `Retailer` (a variation of `SCM`, which its conversations are correlated using key-based approach based on attributes $requestID$ and $responseID$) and `Robostrike` data sets, using the Simple K-means algorithm of the WEKA Data Mining tool (see http://www.cs.waikato.ac.nz/ml/weka/), letting the algorithm automatically decide what is the proper number of cluster in this data set. Note that in the domain of service logs, a cluster correspond to a conversation and, thus, these terms can be used interchangeably. The results showed two things:

- The coherence within the clusters was very large, which is a good indication that WEKA produced good quality clusters.
- There is a big discrepancy between the number of clusters produced by WEKA and those that correspond to real conversations in the data sets.

Looking at the second observation a bit more in details reveals that there is a high overlap among the values of certain attributes in messages of different conversations. This fact alone drives the clustering algorithm to produce much fewer clusters than conversations. In addition, these groups do not necessarily correspond to actual conversations, but e.g., all messages with similar contents are grouped together. Hence, our initial argument that clustering reveals a particular type of conversation that is not potentially useful for message correlation is confirmed.

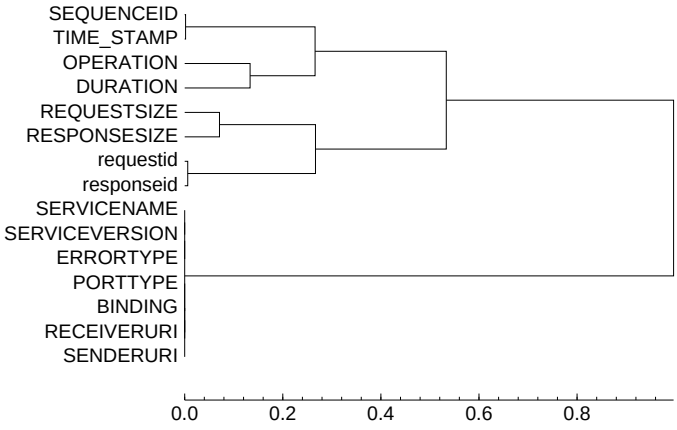



Figure 17: Dendrogram for Retailer dataset

In recent work [2], it has been shown how a categorical clustering algorithm can be used to (a) group the values of a relation such that ones with higher degrees of redundancy in the tuples are placed together, and (b) group the attributes of a relation such that these attributes contain more naturally co-occurring values or, simply, ones with higher degrees of correlation. This type of clustering is sometimes called *vertical* clustering.

This technique is based on information theory and is highly applicable to attributes with categorical values, something that is in accordance with our heuristic assumptions. The final result is a so-called dendrogram that offers a hierarchical representation of the attribute groupings according to the redundancy in their values. Hence, initially each attribute forms its own group. As redundancy is calculated within the attributes, attribute groups start forming by placing together attributes with the highest amount of redundancy. Groupings (or merges) that take place closer to the leaf level of the dendrogam contain very good candidates for (simple) key-based correlators. These will be correlators based on natural co-occurrence of values, *e.g.*, the same customer appearing in the same order over a number of messages.

However, this is only one type of correlation. Our vertical clustering cannot reveal hints on more complex correlation patterns, *e.g.*, reference-based correlation, in which correlator attributes may change from one set of messages to another in a same conversation.

To evaluate the effectiveness of this approach on a key-based correlation scenario, we applied it on the Retailer dataset. The result is given in the dendrogram of Figure 17. In this figure, we observe that attributes whose values naturally co-occur get merged closer to the leaf nodes. For example, $requestID$ and $responseID$ that are key attributes for this dataset are merged together. However, in case of composite key-based rules it is not clear how to derive them from this output.