MailOfMine – Analyzing Mail Messages for Mining Artful Collaborative Processes

Claudio Di Ciccio, Massimo Mecella, Monica Scannapieco, Diego Zardetto, Tiziana Catarci

MailOfMine – Analyzing Mail Messages for Mining Artful Collaborative Processes*

Claudio Di Ciccio[1], Massimo Mecella[1],
Monica Scannapieco[2], Diego Zardetto[2], and Tiziana Catarci[1]

[1] Sapienza – Università di Roma
Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti
Via Ariosto 25, Roma, Italy
{cdc|mecella|catarci}@dis.uniroma1.it
[2] Istituto Nazionale di Statistica
Via Balbo 16, Roma, Italy
{scannapi|zardetto}@istat.it

Abstract. Artful processes are informal processes typically carried out by those people whose work is mental rather than physical (managers, professors, researchers, engineers, etc.), the so called "knowledge workers". In this paper we propose the MailOfMine approach, to automatically build, on top of a collection of email messages, a set of workflow models that represent the artful processes laying behind the knowledge workers activities.

Key words: process mining, email analysis, object matching, visual representation of processes, knowledge workers, artful processes, declarative workflows

1 Introduction

For a long time, formal business processes (e.g., the ones of public administrations, of insurance/financial institutions, etc.) have been the main subject of workflow related research. Informal processes, a.k.a. "artful processes", are conversely carried out by those people whose work is mental rather than physical (managers, professors, researchers, engineers, etc.), the so called "knowledge workers" [1]. With their skills, experience and knowledge, they are used to perform difficult tasks, which require complex, rapid decisions among multiple possible strategies, in order to fulfill specific goals. In contrast to business processes that are formal and standardized, often informal processes are not even written down, let alone defined formally, and can vary from person to person even when those involved are pursuing the same objective. Knowledge workers create informal processes "on the fly" to cope with many of the situations that arise in their daily work. Though informal processes are frequently repeated, they are

* This work has been partly supported by Sapienza – Università di Roma through the grants FARI 2010 and TESTMED, and by the EU Commission through the FP7 project Smart Vortex.

not exactly reproducible even by their originators – since they are not written down – and can not be easily shared either. Their outcomes and information are exchanged very often by means of email conversations, which are a fast, reliable, permanent way of keeping track of the activities that they fulfill. Understanding artful processes involving knowledge workers is becoming crucial in many scenarios. Here we mention some of them:

- *personal information management (PIM)*, i.e., how to organize one's own activities, contacts, etc. through the use of software on laptops and smart devices (iPhones/iPads, smartphones, tablets). Here, inferring artful processes in which a person is involved allows the system to be proactive and thus drive the user through its own tasks (on the basis of the past) [1, 2];
- *information warfare*, especially in supporting anti-crime intelligence agencies: let us suppose that a government bureau is able to access the email account of a suspected person. People planning a crime or an act out of law are used to speak a language of their own to express duties and next moves, where meanings may not match with the common sense. Although, a system might build the processes that lay behind their communications anyway, exposing the activities and the role of the actors. At that point, translating the sense of misused words becomes an easier task for investigators, and allows to infer the criminal activities of the person(s) under suspicion;
- *enterprise engineering*: in design and engineering, it is important to preserve more than just the actual documents making up the product data. Preserving the "soft knowledge" of the overall process (the so-called product life-cycle) is of critical importance for knowledge-heavy industries. Hence, the idea here is to take to the future not only the designs, but also the knowledge about processes, decision making, and people involved [3, 4, 5].

The objective of the approach, proposed here, is to automatically build a set of workflow models that represent the artful processes laying behind the knowledge workers activities, on top of a collection of email messages. There are many advantages out of this work. First of all, the unspecified agile processes that are autonomously used become formalized: since such models are not defined *a priori* by experts but rather inferred from real-life scenarios that actually took place, they are guaranteed to respect the true executions (often Business Process Management tools are used to show the discrepancy between the supposed and the concrete workflows). Moreover, such models can be shared, compared, preserved, so that the best practices might be put in evidence from the community of knowledge workers, to the whole business benefit. Finally, an analysis over such processes can be done, so that bottlenecks and delays in actual executions can be found out.

The approach we want to pursue involves many research fields at a time, each one concerning a phase of the overall processing. We make use of *object matching* algorithms to obtain clusters of related email conversations. Every cluster is subsequently treated by *text mining information extraction* procedures, in order to find out which tasks email messages are about. *Process mining* is used to

abstract process models representing the workflows, which the sets of subsumed tasks were considered traces of.

Our approach is named MailOfMine.

Here we also discuss how we addressed the challenge of showing artful processes to users, that are knowledge workers mainly, through a graphical interface which is flexible, functional and easy to understand. It is designed to be validated and further improved in collaboration with them. A user-centered methodology is going to be applied, not only to the development of the graphical interface, but also to the specification of the visual notation that describes the processes.

The remainder of the paper is organized as follows. Section 2 describes the adopted process model, and Section 3 presents the approach. Section 4 focuses on the techniques adopted for email clustering and their validation. Section 5 presents the visual notation for artful processes and the proposed user interface of the tool. Section 6 discusses and compares the relevant related work. Finally, Section 7 concludes the paper and outlines the future activities.

2 The Process Model

In this section, we present the process model adopted for describing the mined artful processes. After a clarifying example, we will discuss technical and theoretical implications of our assumptions.

2.1 Definitions

Definition 1 (Actor). *An* actor *is the subject directly or indirectly taking part in the progress of a work. She is called* contributor *when her collaboration is either proven by the participation to the collaborative communications or by the production of outcomes. Otherwise, she is named* spectator.

The spectator is the person who receives messages but does not reply, i.e., is among the recipients but is never a sender during the discussion. Nonetheless, her presence is perhaps fundamental, since she is a stakeholder, or a manager controlling the thread though having no reason to intervene ("no news, good news").

Definition 2 (Task). *A* task *is an elementary unit of work. Each task is connected to* (i) *its expected* duration, (ii) *zero or more* outcome*s, and* (iii) *one or more* actor*s. A task is called* productive *when linked to one or more outcomes. Otherwise, it is named* clarifying.

Clarifying tasks are not less important than productive ones. Often they are crucial to define, e.g., aim and methodology of the further steps. The outcome is whatever adds a bit of information to the exchanged knowledge among actors. In our context we simplify the definition by considering *document oriented* outcomes, i.e., we consider as new products of the tasks only those documents which are attached to the email body.

Definition 3 (Activity). *An* activity *is a collection of tasks or (recursively) other activities.*

Definition 4 (Key Part). *A* key part *is each* unique *piece of text belonging to the email messages exchanged in a communication. Thus, given a collection of duplicated pieces of text (coming from the same or different email messages in the thread), just a single representative is selected as key part.*

For instance, HTML signatures used by the sender of the email, quotations of previous email messages used in replies, etc., are all examples of redundant information that may appear in a communication thread, but will be filtered out by means of the *key part* concept. On the other hand, any piece of text not appearing in any other email message in the discussion thread will be interpreted as *key part*.

Definition 5 (Indicium). *An* Indicium *is any communication thread, or part of it, attesting the execution of a task, an activity, or a process instance.*

Indicia are key parts sets (or sets of key parts sets) containing any evidence that a task, or an activity, or a process instance, has been performed or is being performed.

Definition 6 (Process Scheme (Process) and Process Describing Grammar ($\mathcal{PDG}$)). *A* process scheme *(or* process *for short) is a semi-structured set of activities, where the semi-structuring connective tissue is represented by the set of constraints stating the interleaving rules among activities or tasks. Such a set of constraints, formulated on top of activities and tasks, is named* Process Describing Grammar ($\mathcal{PDG}$).

Constraints do not force the tasks to follow a tight sequence, but rather leave them the flexibility to follow different paths, to terminate the enactment, though respecting a set of rules that avoid illegal or non-consistent states in the execution.

2.2 On the Process Describing Grammar

Let us consider every possible task [2] in a process as a character of an alphabet. Activities are thus structured actions which are either allowed or not to be executed in a process, as characters may or may not appear inside a string. The execution of some activities can imply others to be implied, at any point in time, as well as production rules enable the modification of the string expression in case a given sequence of characters is reached.

In this paper, we argue that each constraint useful to define declarative models for artful processes is expressible through regular grammars [6]. Regular grammars are recognizable through Finite State Automata (FSA) [7] (either

[2] In the following, we focus on task identifiers, omitting, for sake of readability, details about duration, actors and outcomes. Hence, we identify tasks by their names only.

deterministic or non-deterministic [8]). Constraints on the process always hold and must be respected altogether at each point in time. This means that the intersection of all the constraints must always hold. In other words, the FSA recognizing the correct traces for processes (i.e., accepting the valid strings) is the intersection of all the FSAs composing set of constraints (it is known that such grammars are closed to the operation of intersection [9]). Thus, we consider each task as a terminal character in the $\mathcal{PDG}$. Each constraint on tasks is a $\mathcal{PDG}$ itself. An activity is thus a $\mathcal{PDG}$ built as the intersection of all of the constraints' $\mathcal{PDG}$s.

In order to define the process scheme, we shift to regular expressions, which are an equivalent way to describe regular grammars (and accepting FSAs as well). Regular expressions are closed to their operators, thus we are able to recursively define activities as the intersection of constraints on activities. The process scheme, in turn, is the intersection of constraints on activities.

We finally summarize the concepts introduced so far, from a hierarchical recursive point of view. Tasks are terminal characters, building blocks of constraints on tasks. Constraints are regular expressions, equivalent to regular grammars. Activities are regular expressions which are equivalent to the intersection of the constraints' regular grammars. Constraints can be formulated on top of activities, being regular expressions themselves. The process scheme is the intersection of constraints defined on top of activities.

Being the regular grammars expressible by means of regular expressions, we can consider each constraint as a regular expression (corresponding to an accepting FSA). Moving to this domain, we can compose more complex regular expressions as if each constraint was a building block and thus compose them.

Here, we adopt the Declare [10] taxonomy of constraints as the basic language for defining artful processes in a declarative way. But whereas in Declare constraints are translated into LTL formulas, we express each constraint through regular expressions, as shown in Table 1.

For sake of brevity, there we used the POSIX standard shortcuts. Therefore, in addition to the known Kleene star (`*`), concatenation and alternation (`|`) operators, we make use here of *(i)* the `.` and `[^`x`]` shortcuts for respectively matching any character in the alphabet, or any character but x, and *(ii)* the `+` and `?` operators for respectively matching from one to many, or zero to one, occurrences of the preceding expression.

The $Existence(m, a)$ constraint imposes `a` to appear at least m times in the trace. The $Absence(n, a)$ constraint holds if `a` occurs at most $n - 1$ times in the trace. $Init(a)$ makes each trace start with `a`. $RespondedExistence(a, b)$ holds if, whenever `a` is read, `b` was already read or is going to be read (i.e., no matter if before or afterwards). Instead, $Reponse(a, b)$ enforces it by forcing a `b` to appear after `a`, if `a` was read. $Precedence(a, b)$ forces `b` to occur after `a` as well, but the condition to be verified is that `b` was read - namely, you can not have any `b` if you did not read an `a` before. $AlternateResponse(a, b)$ and $AlternatePrecedence(a, b)$ both strengthen respectively $Response(a, b)$ and $Precedence(a, b)$. The "alternation" is in that

Constraint	Regular expression	Example
Existence constraints		
$Existence(m, a)$	`[^a]*(a[^a]*){m,}[^a]*`	bc**aa**c for $m = 2$
$Absence(n, a)$	`[^a]*(a[^a]*){0,n}[^a]*`	bc**aa**c for $n = 3$
$Init(a)$	`a.*`	**a**ccbbbaba
Relation constraints		
$RespondedExistence(a, b)$	`[^a]*((a.*b)|(b.*a))*[^a]*`	**bca**accbbb**a**b**a**
$Response(a, b)$	`[^a]*(a.*b)*[^a]*`	bc**a**acc**bbba**b
$AlternateResponse(a, b)$	`[^a]*(a[^a]*b)*[^a]*`	bc**a**cc**b**bb**ab**
$ChainResponse(a, b)$	`[^a]*(ab[^a^b]*)*[^a]*`	bc**ab**bb**ab**
$Precedence(a, b)$	`[^b]*(a.*b)*[^b]*`	c**a**acc**b**bb**a**b**a**
$AlternatePrecedence(a, b)$	`[^b]*(a[^b]*b)*[^b]*`	c**a**acc**ba**b**a**
$ChainPrecedence(a, b)$	`[^b]*(ab[^a^b]*)*[^b]*`	c**ab**a**b**a
$CoExistence(a, b)$	`[^a^b]*((a.*b)|(b.*a))*[^a^b]*`	**bca**ccbbb**a**b**a**
$Succession(a, b)$	`[^a^b]*(a.*b)*[^a^b]*`	c**a**acc**b**bb**ab**
$AlternateSuccession(a, b)$	`[^a^b]*(a[^a^b]*b)*[^a^b]*`	c**a**cc**bab**
$ChainSuccession(a, b)$	`[^a^b]*(ab[^a^b]*)*[^a^b]*`	c**abab**
Negative relation constraints		
$NotChainSuccession(a, b)$	`[^a]*(a[^a^b])*[^a]*`	bc**a**acc**b**bbb**a**
$NotSuccession(a, b)$	`[^a]*(a[^b]*)*[^a^b]*`	bc**a**acc**a**
$NotCoExistence(a, b)$	`[^a^b]*((a[^b]*)|(b[^a]*))?`	c**a**acc**a**

Table 1: Semantics of Declare constraints as regular expressions

Constraint	Regular expression	Example
Existence constraints		
$Participation(a) \equiv Existence(1, a)$	`[^a]*(a[^a]*)+[^a]*`	bc**a**ac
$Unique(a) \equiv Absence(2, a)$	`[^a]*(a)?[^a]*`	bc**a**c
$End(a)$	`.*a`	bcaaccbbbab**a**

Table 2: Additional constraints in MailOfMine

you can not have two a (b) in a row before b (after a). $ChainResponse(a, b)$ and $ChainPrecedence(a, b)$, in turn, specialize $AlternateResponse(a, b)$ and $AlternatePrecedence(a, b)$, both declaring that no other symbol can occur between a and b. The difference between the two is in that the former is verified for each occurrence of a (b must immediately follow), the latter for each occurrence of b (a must immediately precede). $CoExistence(a, b)$ holds if both $RespondedExistence(a, b)$ and $RespondedExistence(b, a)$ hold. $Succession(a, b)$ is valid if $Response(a, b)$ and $Precedence(a, b)$ are verified. The same holds with $AlternateSuccession(a, b)$, equivalent to the conjunction of $AlternateResponse(a, b)$ and $AlternatePrecedence(a, b)$, and with $ChainSuccession(a, b)$, w.r.t. $ChainResponse(a, b)$ and $ChainPrecedence(a, b)$. $NotChainSuccession(a, b)$ expresses the impossibility for b to occur immediately after a. $NotSuccession(a, b)$ generalizes the previous by imposing that, if a is read, no other b can be read until the end of

the trace. $NotCoExistence(a, b)$ is even more restrictive: if a appears, not any b can be in the same trace.

In addition to the constraints above, we also consider the ones described in Table 2. Taking inspiration from the relational data model cardinality constraints, we call *(i)* $Participation(a)$ the $Existence(m, a)$ constraint for $m = 1$ and *(ii)* $Uniqueness(a)$ the $Absence(n, a)$ constraint for $n = 2$, since the former states that a must appear at least once in the trace, whereas the latter causes a to occur no more than once. $End(a)$ is the dual of $Init(a)$, in the sense that it constrain each string to end with a. Beware that $End(a)$ would be clueless in a LTL interpretation, since LTL is thought to express temporal logic formulae over infinite traces. On the contrary, it makes perfectly sense to have a concluding task for a finite process, expressed by means of a regular automaton.

We recall here that the graphical syntax proposed for these constraints is presented in Section 5.

2.3 An Example

Let us suppose to have an email archive, containing various process instances indicia, and to focus specifically on the planning of a new meeting for a research project. We suppose to execute the overall technique explained in Section 3, and we report the possible result, starting from the list of tasks in activities (Process Description 1).

Process Description 1 Activities and tasks list

Activity: $\langle Flight \rangle$
  Task: $boFl$ ("bookFlight"): Productive
    Actors: {You: Contributor, $FlightCompany$: Contributor}
    Duration: 2 hrs.
Activity: $\langle Agenda \rangle$
  Task: $prAg$ ("proposeAgenda"): Productive
    Actors: {You: Contributor, $Community$: Spectator}
    Duration: 4 dd.
  Task: $rqAg$ ("requestAgenda"): Clarifying
    Actors: {$Participant$: Contributor, $Community$: Spectator}
    Duration: $\perp$
  Task: $coAg$ ("commentAgenda"): Clarifying
    Actors: {$Participant$: Contributor, $Community$: Spectator}
    Duration: $\perp$
  Task: $cnAg$ ("confirmAgenda"): Productive
    Actors: {You: Contributor, $Community$: Spectator}
    Duration: 2 dd.
Activity: $\langle Accommodation \rangle$
  Task: $esHo$ ("establishHotel"): Productive
    Actors: {$Organizer$: Contributor, $Community$: Spectator}
    Duration: $\perp$
  Task: $boHo$ ("bookHotel"): Productive
    Actors: {You: Contributor, $Hotel$: Contributor}
    Duration: 2 dd.

The "bookFlight" task is supposed to be retrieved by confirmation email messages received by the user when the booking is completed. The $\langle Flight \rangle$ activity is composed by this task only. $\langle Agenda \rangle$ is supposed to be slightly more

complex instead. We suppose that a final agenda will be committed ("confirmAgenda") after that requests for a new proposal ("requestAgenda"), proposals themselves ("proposeAgenda") and comments ("commentAgenda") have been circulated in the group (*Community*, as identified in the Actors list). Finally, $\langle Accommodation \rangle$ is the activity related to the reservation of rooms, in a hotel located where the meeting will take place (or nearby). Beyond the confirmation email messages received by the user when the booking is completed (indicia for the "bookHotel" task), the "establishHotel" stems from the email messages informing about the hotel arranged by the meeting organizers.

The reader may notice that sometimes *You* appears among the Actors. We suppose it to be the special identifier adopted whenever the owner of the email archive is a Contributor. In case she is only a Spectator (see, e.g., "requestAgenda"), such an information is not reported, since it is redundant (if the owner were not among the recipients, there should have been no way to access email messages related to that task). The usage of $\perp$ as Duration stands for an unknown value: whenever there is only one indicium related to a task (i.e., only one email message proving its execution, for each activity indicium), no inference about Duration can be performed.

The aforementioned tasks and activities are bound to the following constraints. In order to express them, we use the base set introduced before in this Section, starting with the *existence* constraints of Process Description 2.

Process Description 2 Existence constraints on the example tasks and activities

Activity:	$\langle Flight \rangle : [0, *]$	
Task:	$boFl : [1, *]$	i.e., $\{Participation(boFl)\}$
Activity:	$\langle Agenda \rangle : [1, 1]$	i.e., $\{Participation(\langle Agenda \rangle), Uniqueness(\langle Agenda \rangle)\}$
Task:	$prAg : [0, *]$	
Task:	$rqAg : [0, *]$	
Task:	$coAg : [0, *]$	
	$Init(coAg)$	
Task:	$cnAg : [1, 1]$	i.e., $\{Participation(cnAg), Uniqueness(cnAg)\}$
Activity:	$\langle Accommodation \rangle : [0, 1]$	i.e., $\{Uniqueness(\langle Accommodation \rangle)\}$
Task:	$esHo : [0, 1]$	i.e., $\{Uniqueness(esHo)\}$
Task:	$boHo : [1, 1]$	i.e., $\{Participation(cnAg), Uniqueness(boHo))\}$

In Process Description 3 we report the relation constraints holding in this example process.

Process Description 3 Relation constraints on the example tasks and activities

Activity: $\langle Agenda \rangle$
  $Response(rqAg, prAg)$
  $RespondedExistence(coAg, prAg)$
  $Succession(prAg, cnAg)$
$CoExistence(\langle Flight \rangle, \langle Accommodation \rangle)$
$RespondedExistence(\langle Agenda \rangle, \langle Accommodation \rangle)$

3 The MailOfMine Approach

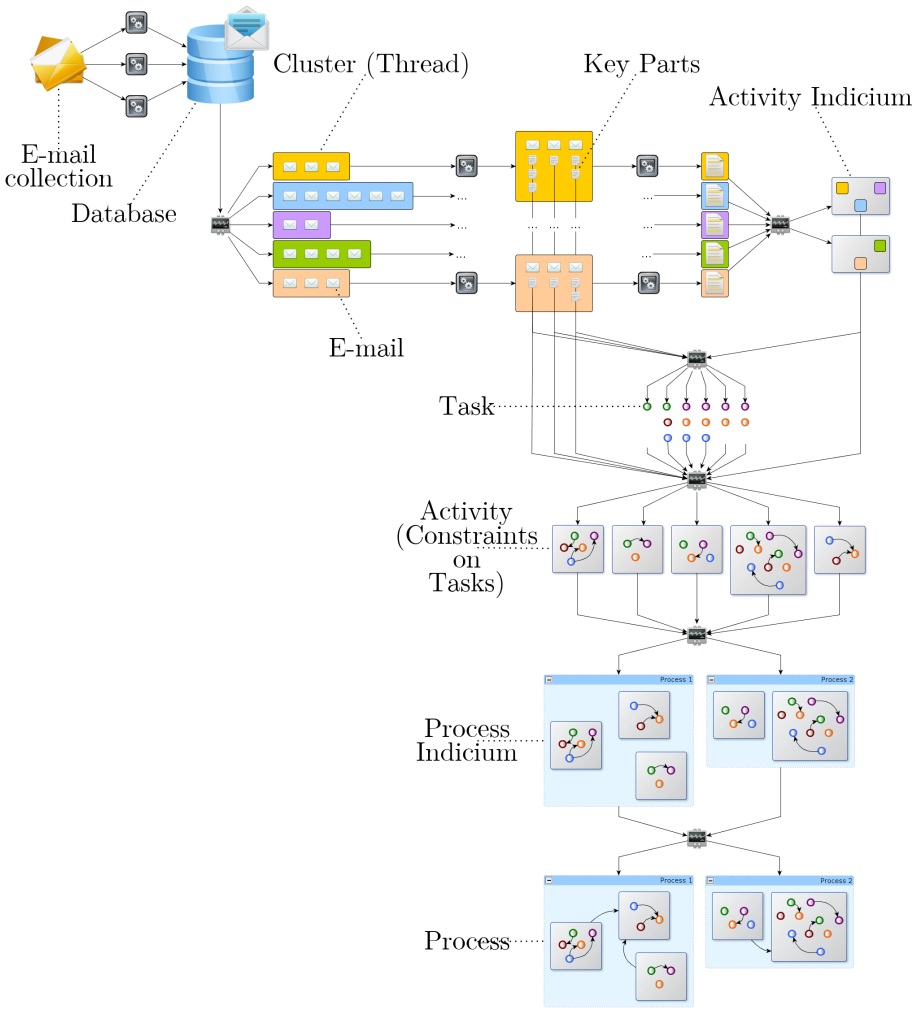


Fig. 1: The MailOfMine approach

The MailOfMine approach (and the tool we are currently developing) adopts a modular architecture, the components of which allow to incrementally refine the mining process, as in Figure 1. We describe it into three parts: *(i)* the preliminary steps, from the retrieval of email messages to the reconstruction of communication threads; *(ii)* the extraction of key parts, the activities and tasks

indicia detection, and the tasks definition; *(iii)* the final steps, from the activities definition to the final mined process extraction.

Section 5 describes the design rationale for the graphical representation of the mined process.

3.1 Preliminary Steps and Email Clustering

First of all, we need to extract email messages out of the given archive(s). Archives are compliant to different standards, according to the email client in use, hence the component for reading the messages is intended to be plug-in based: each plug-in is a component able to access multiple standalone applications' archive formats (e.g., Mozilla Thunderbird, Microsoft Outlook, etc.) and/or online providers (e.g., Gmail, Yahoo! Mail, etc.). The outcome is the population of a database, on the basis of which all the subsequent steps are carried out.

The first of them is the clustering of retrieved messages into extended communication threads, i.e., flows of messages which are related to each other. The technique that is used to guess such a connection is based not only on the Subject field (e.g., looking at "Fwd:" or "Re:" prefixes) and the SMTP headers (e.g., reading the "In-Reply-To" field), but on the application of a more complex object matching decision method. Such indicators, though likely trustworthy, might be misleading: *(i)* in the everyday life, it is a common attitude to reply to old messages for talking about new topics, which may have nothing to do with the previous one; *(ii)* conversely, it may happen that a new message is related to a previous, though written and sent as if it were a new one. The technique is detailed further in Section 4.

Once the communication threads are recognized, we can assume them all as activity indicia candidates.

After the clustering phase, messages are analyzed in order to identify key parts. I.e., email messages are cleaned up from signatures and quotations citing some text already written in another message within the thread, by the usage of a combination of the techniques described in [11] and [12]. The key parts are the text remaining in bodies and subjects once such filtering operation is performed.

3.2 Identifying Activities

Once messages and key parts in threads are gathered, MailOfMine can build activity indicia as the concatenation of all the key parts (artificial text). Then, the clustering algorithm is used again, this time to identify the matches between activity indicia. E.g., let us suppose to have *(i)* a thread T_{del41} related to the writing of Deliverable 4.1 for a research project, *(ii)* another thread T_{del52} related to the writing of Deliverable 5.2 for the same research project, and, finally, *(iii)* a T_{air} dealing with an airplane flight booking, for the next review meeting. Thus, the algorithm is expected to cluster as follows: $\{T_{del41}, T_{del52}\}, \{T_{air}\}$. Its output represents the set of activities.

By taking into account the set of activities and the key parts (task indicia candidates), the clustering algorithm checks for matching key parts, identifying them

as tasks. E.g., let us suppose to have *(i)* a key part $k_{airData}$ specifying the booked airplane data, *(ii)* another key part $k_{airBook}$ containing the confirmation of the booking, and, finally, *(iii)* a k_{del41}: "Please find attached Deliverable 4.1". Thus, the algorithm is expected to cluster as follows: $\{k_{airData}, k_{airBook}\}, \{k_{del41}\}$.

Hence, each set is a task indicium for one task. The analysis, though, does not terminate here. In order to understand whether the task under investigation is clarifying or productive, MailOfMine checks *(i)* if any document-oriented outcome in the original email messages was attached; *(ii)* if key parts contain Speech Acts [13], according to the technique proposed by [14]. If at least one of the listed conditions holds, it is productive. Otherwise, it is considered as clarifying. The detection of Speech Acts is supposed to be assisted by the experts, who are required to provide a dictionary of keywords of their domain field, as in [14]. For further information on the relation between Speech Acts and workflows, see [15]. The task name will be given as a rule of thumb by the conjunction of verbs and dictionary keywords composing the Speech Acts, although the user will be able to customize it at her wish.

3.3 Mining the Process

For each thread (activity indicium), tasks are put in sequences respecting the ordering of the task indicia (key parts with a Speech Act), given by the timestamp of the email they belong to. Tasks appear in many traces then, being the indicia potentially found in different threads. Thus, threads with email messages are turned into traces of enacted tasks. MailOfMine searches for execution constraints between tasks, stemmed from these traces, though an algorithm used in purpose, named MINERful (presented in [16] and detailed in [17]). The intersection of the mined constraints on tasks constitutes the $\mathcal{PDG}$ of activities.

Once activities and tasks are recognized, a supervised learning process takes place, in order to cluster activities into processes. This step cannot be fully performed by the system, since no linguistic connection could exist among related activities. E.g., the activity of drawing slides and the reservation of the airplane could sound completely apart, though those slides could be the material to present at a review meeting, hold in the city that the airplane was booked to reach.

The activity grammar would not be exposed to the user, of course. Instead, the threads are shown as witnesses for the activity; hence, the user associates part of them to different processes, and the learner associates all the related activities to processes. The choice of the name to be assigned to activities and processes is left to the user.

Once processes are identified, MailOfMine performs the second step for the construction of the $\mathcal{PDG}$, i.e., the mining of production rules among activities inside the same proces. For each activity indicium (thread), a timeframe is defined as the window between the timestamps of the first and the last email in the thread. Thus, traces are composed by ordering activity indicia belonging to the same process, on the basis of their timeframe. These traces are the input for MINERful to run again.

4 Email Clustering

In this section we describe how we tackle the problem of performing a Similarity Clustering (SC) of email messages. The purpose of such a clustering step is to identify groups of related email messages to be used for finding tasks that compose process activities. To face the SC problem, we exploit a previously proposed Object Matching (OM) algorithm [18], providing ad-hoc extensions for the task at hand. Here we briefly illustrate the rationale for adopting an OM algorithm to cluster email messages. From an abstract point of view, OM and SC share the common objective of classifying data-object pairs according to an hidden (i.e. not self-evident) property; for OM, the relevant hidden property to be discovered is if two objects *"do represent – or not – the same real-world entity"*, whereas, for SC, one needs to asses if two objects *"do belong – or not – to the same group"*. Moreover, in order to *infer* the class the pairs belong to, both OM and SC rely on pairwise distance (or, equivalently, similarity) measures. With respect to the choice of the specific OM algorithm [18], we point out that it relies on its fully automated nature and on its independence on the specific data object representation. Lastly, we remark that the statistical properties at the basis of the chosen algorithm are perfectly valid for the SC problem.

We decided to adopt an Object Matching technique, rather than classical Clustering, due to the fact that it extends the Record Linkage field (see Section 6). Coming from the database domain, RL produced dedicated frameworks and tools to cope with textual issues, over the years. For instance, one of their mandatory requirements was to be resilient to typos. Clustering techniques are focused on the similarity grouping of numerical entities, such as points or vectors in N-dimensional spaces, instead. Therefore, we exploit an algorithmic machinery developed to deal with text-related challenges.

We implemented our Similarity Clustering as a four steps procedure:

1. in the first step, we chose a representation format for email messages, in order to translate them into processable data objects;
2. the second step consisted of the definition of a specific distance metric to compare email objects;
3. as a third step, we ran a decision algorithm whose output was a set of email pairs declared as "matches", i.e. – in the present context – belonging to the same cluster;
4. the fourth and last step implied the application of a function performing a transitive closure among "match" pairs in order to build email clusters.

This procedure is detailed in the following together with some test results.

4.1 Email Object Representation and Distance Metric Definition

In order to compare email objects we first defined a representation format for each email message. Specifically, we considered each email as a record consisting of: *(i)* some header fields, *(ii)* the body and *(iii)* the names of attached files (if present).

We observe that, according to RFC 5322 and RFC 3864 (`http://www.ietf.org/rfc.html`), each message has exactly one header, which is structured into fields. Some of these fields are mandatory (such as the *message_id*), others are instead optional but indicated as common. Among such common header fields, we took into account the ones listed in the following; for each of them we also specify the chosen representation format:

- `emailIdentifier`: this is a string associated to each email that permits its unique identification.
- `Sender`: email address of the sender represented as a string.
- `Receivers`: email addresses represented as strings and concatenated into a unique string.
- `Subject`: represented as one single string.

Having as objective to build clusters consisting of email exchanged to accomplish a specific task, we found reasonable to merge `Sender` and `Receivers` fields into one single field `SenderReceivers`. This is because the clusters we want to determine are, in a sense, *invariant* with respect to the email exchange direction, as they are instead focused on the actions performed by means of such email exchanges.

As far as the body, we chose to consider it as a piece of free text and we represented it as one single string.

Finally, email attachments were taken into account by considering their names. More specifically, we represented attachments by a unique string, `AttachmentsNames`, resulting from the concatenation of the names of all the present attachment files [3].

On the basis of the representation choices described above, all the email fields but the body could be compared by means of traditional string metric distances (see [19] for a survey). In our experiments, we used the Levenshtein distance for these fields.

The body is an exception as it is a piece of free text rather than a (more or less) *structured* text field like the others[4]. The set of the bodies in an email collection is considered as a corpus of documents. As a first step, each body is regarded as a bag of words ("terms"). Then, we build a Term-Document matrix. To each term of a body a weight was assigned, based on the traditional TF-IDF metric [20]. Each body is in this way represented by a vector of a vector space and hence easily comparable to the other bodies. To such a scope, we use the cosine distance function. The distance so obtained will be referred as VSM distance (as it is based on the Vector Space Model [21]). TF-IDF was considered appropriate because of its ability to increase the importance of a term proportionally to the

[3] Given the disparate type of possible attachment files (e.g. video, images, etc.) a generalized approach that exploits attachment contents has not been considered as viable.

[4] Though the subject can contain more than one word, we decided to consider it too as a structured field. Indeed, since typically subjects are not long unstructured texts, text mining techniques are unnecessary, or even not suitable, for their comparison.

term frequency inside the document, while penalizing terms that are found to be very common in the corpus.

Our choices for email representation and the distance functions used for each field are summarized in Table 3, below.

Table 3: Email fields used for comparison and related distance functions

Field	**Distance**	**Field**	**Distance**
`Subject`	Levenshtein	`SenderReceivers`	Levenshtein
`AttachmentsNames`	Levenshtein	`Body`	VSM

4.2 Experimental Results

The testbed for our SC application consisted of two experiments. The first one worked on a small collection of 101 email messages, in order to be able to conduct a manual evaluation of the quality of the obtained clustering results. The second experiment worked instead on a larger collection of 1337 email messages, including the previous, in order to test the practical feasibility of our approach with a realistic mailbox dimension. The analyzed set of email messages represents the collection of messages exchanged during the development of a European research project that one of the authors was involved in. The smaller collection was the part of email messages sent or received in a given frame of the project lifetime.

In the first experiment, the 101 input email messages generated 5050 email-pairs. Our decision algorithm declared 98 pairs to belong to the same group. The transitive closure performed on the set of such 98 "matching" email-pairs determined 21 non overlapping and non trivial (that is containing at least 2 email messages) clusters: these were the final output produced by our SC algorithm. The output clusters size ranged from 2 to 11 email messages and the email messages involved in the clusters are found to be 68 out of 101.

To assess the quality of the obtained result, we manually analyzed the input email messages and the output clusters: we identified only 1 false positive cluster, whereas no false negatives were found. The false positive cluster involved two email messages, which happened to have the same sender and receivers, and almost completely overlapping short bodies. The very encouraging information is that no false negative was found, thus it seems there is no need to re-iterate it over times to recover from a possible information loss.

Out of the 893116 pairs generated by the 1337 input email messages of the second test, our decision algorithm declared 2974 email-pairs to belong to the same group. The transitive closure performed on the set of such 2974 "matching" email pairs determined 230 different clusters. The output clusters size ranged from 2 to 39 email with a mean of 4.6, and the email messages involved in the clusters were found to be 1060 out of the original 1337.

Given the size of our second experiment, building a "gold-standard" clustering solution by manual inspection was clearly unmanageable. Therefore, we

were not able to evaluate the quality of our clustering results *directly*, that is in terms of false positives and false negatives. As an alternative, we rather tried to get a feeling of the goodness of our reconstructed email clusters as a mean for identifying tasks. To this end, we first asked the owner of the mailbox archive to provide us a small user-generated dictionary of relevant and common terms in the research-projects domain (like, e.g., "deliverable", "workpackage", "meeting", ...); the dictionary we received had 56 entries. Next, we treated the bodies of the email messages belonging to each identified cluster as a corpus of documents and assigned to each word inside such corpora its TF-IDF weight. Then we selected, for each cluster, the top-30 TF-IDF weighted words as representatives of the related e-mail conversation. We manually stemmed them to singular nouns and infinitive verbs (e.g., "deliverables" was turned into "deliverable" as "asks" into "ask"). Lastly, we applied an intersection operation between each of the 30-word-sets per cluster and the 56 entries of the user-generated dictionary. The outcome of such an exercise seems quite encouraging, as 214 of the 230 clusters (i.e., 93%) turned out to be represented by at least one of the domain-specific dictionary words.

5 Process Visualization

The literature dealing with the representation of processes typically aims at visualizing the process all at once, by means of diagrams that show the complete grid of interconnections among activities. Here we propose a change in the viewpoint. We want to model artful processes as a collection of constraints, through the declarative approach. Being highly flexible, this kind of representation does not necessarily impose a pre-defined strict order on activities, neither explicit nor implicit. For instance, one can state that a task a implies the execution of another task b afterwards (see Section 2.3), with no specification provided if a is not performed, meaning that b can be done or not during the process instance run. In other words, the process schema itself can change according to the things that may have happened before. This is why we do not consider as the best suitable solution adopting a static graph-based global representation alone, on one hand: a local view should work better in conjunction with it. On the other hand, no knowledge worker is expected to be able to read and understand the process by reading the list of regular-expression based constraints: a graphical representation, easy to understand at a first glimpse, must be used.

The process schema and the running processes are respectively modeled through *(i)* a set of diagrams, representing constraints on workflows (static view: Section 5.1) and *(ii)* an interactive evolutionary graphical representation for the visualization of running instances (dynamic view: Section 5.2). Furthermore, we propose two complementary views on constraints: *(i)* a local view focusing on one activity at a time and *(ii)* a global view providing a bird-eye sketch of the whole process schema.

As activities are basically collections of tasks (thus, they can be single tasks too), which have to be compliant with the same constraints as tasks, in the

following we will consider tasks only, for sake of simplicity. The same statements remain valid for activities, though. For sake of simplicity and readability of figures, we adopt one-character identifiers for tasks, though they are going to adhere in practice to the naming rules described in Section 3.2, as the graphical representation of artful processes presented here is intended to be the core of the user interface for the visualization of the mined processes in MailOfMine.

5.1 Process Schema

The local view. It is very hard to show a process schema all at once and keep it easily readable, due to the high flexibility of the declarative representation. Thus, given that the declarative approach is based on constraints, we collect all of those related to every single task, i.e., where the task (e.g., e) is either *(i)* directly implied (e.g., if d is done, then e must be done), or *(ii)* directly implying (e.g., if e is done, no matter when, f was done before or must be done in the future). The *directly* adverb is used due to the need not to make things too much complicated and to follow the idea of having a local view only. For instance, the process is such that if d is done, then e must be performed; moreover, the enactment of c implies that d can not be done further. If we look at the constraints directly affecting e, the latter rule is not taken into account. In fact, if c is not performed, nothing is imposed on d. This is an example providing a hint on the rationale: for sake of readability, we want to avoid the confusion coming from too many cross-implications to consider at a time.

The representation of relation constraints is based on three main degrees of freedom, namely *(i)* time, *(ii)* implication, *(iii)* repeatability. The time is considered here as a discrete ordered set of steps the tasks can take place in. We ideally consider each task as spending a single unit in this conception of time. The notion of implication is a based on two values (implying, implied). The repeatability is given by the specification of one among four values, standing for the number of times a task can be consequently fulfilled: *(i)* zero, one or more times; *(ii)* zero, or one time; *(iii)* exactly once; *(iv)* zero times.

Our graphical notation represents time and implication as the coordinates of a bidimensional drawing, where time is on the ordinates. This ideal y axis divides the plane space into two separate regions: one for each value of the implication dimension (implying, implied), on the abscissae. The x axis divides the plane space into two regions: upwards, what can (or can not) happen *before* the task is executed, and, downwards, what can (or can not) happen after. On the origin of this chart, inspired to the cartesian coordinate system, we put the task under examination. The y axis is oriented towards the bottom, in order to follow the reading directionality. For the same reason, the implication relation order flows from the left to the right. Of course, the orientation of the axes can change according to the localization of the software running: e.g., users from Arabic countries might prefer a mirrored version, where the implied tasks are on the left, the implying on the right.

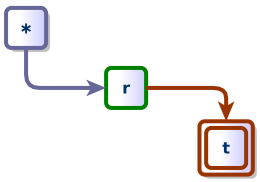
The repeatability is expressed by the thickness of the boundaries around the boxes representing tasks: dashed for tasks that can be done zero, one or more

times, solid for zero or one times, double-line for exactly one time. The task box turns into a cross shape when repeatability is zero. The repeatability is referred to the quadrant the box appears in. For instance, u must appear once either before or after e took place. We recall here that the scope of repeatability, as all of the other degrees of freedom, is not extended to the whole process instance existence, but only for what concerns the time surrounding the single task under analysis.

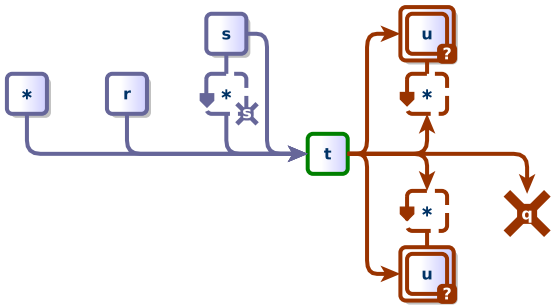
For sake of readability, we do not explicitly mention every possible task the process can be composed of, on the graph. Instead, we render only such tasks that are interested in focused constraints, so to address the potential problem of too many nodes and arcs connecting one another in a scarcely readable spaghetti-like diagram. Though, visualizing the tasks involved in constraints only, might look like a way to force the actor to execute nothing else than the ones that are shown. On the contrary, declarative models allow to do more: roughly speaking, what is not mentioned, is possible. Thus, we make use of a wildcard ($*$) not intended as "every task" in the usual all-comprehensive form, but in the typical human conception: "any task", where it is understood that the other rules remain valid (e.g., if it is stated that q can not be executed after t, a $*$ after t means "any task, except q"). Examples of the diagrams are in Figure 2. The constraints depicted here are described in Section 2.

The graphical notation is enforced by arrows, easing the user to go across the flow of tasks, from the implying before to the implied afterwards. Colors are used for sake of readability and comprehensibility, as additional arrows making a loop on zero-one-more-repeatable tasks, though the idea is that such diagrams must be kept easy to be sketched by a pen, as well.

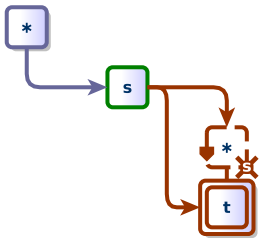
Figure 2a shows the only constraint pertaining r in an imaginary process, namely $ChainSuccession(r, t)$. It states that immediately after (i.e., below, on the diagram) the implying r (on the center) you must (double line) execute t, as an implication (on the right). Nothing is directly told about r as an implied task in a constraint: thus, a $*$ wildcard is put on the left of r. Then, Figure 2c focuses on s, which is the implying task for the $AlternateResponse(s, t)$ constraint: i.e., if s is executed, you can either perform t (see the direct arc in the fork on the right of s) or optionally (dashed line) perform any other task ($*$) but s (put inside the cross at the right bottom corner of the dashed box) until you do t. t must be executed in any case: this is why the line bounding the box is double. Figure 2b details the constraints that concern t. On the right side of the box with t, the $RespondedExistence(t, u)$ and $NotSuccession(t, q)$ are drawn. The former causes the arcs to fork both upwards and downwards, due to the fact that the $RespondedExistence$ constraints do not specify whether the implied task must be done before or after the implying. The question mark put on the right bottom corner is used to enforce this concept of optionality, together with the double line recalling that the execution of t is bound to the execution of u – i.e., no matter if beforehand or afterwards, u must be enacted. The $NotSuccession(t, q)$ is represented by the cross that the q identifier is inscribed in, meaning that q can not be executed after (below) t. On the left of t



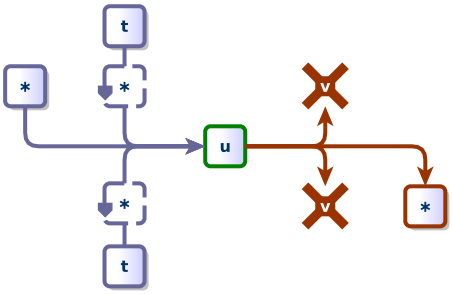
(a) $\{ChainSuccession(\mathbf{r}, t)\}$



(b) $\{ChainSuccession(r, \mathbf{t}), AlternateResponse(s, \mathbf{t}), RespondedExistence(\mathbf{t}, u), NotSuccession(\mathbf{t}, q)\}$



(c) $\{AlternateResponse(\mathbf{s}, t)\}$



(d) $\{RespondedExistence(t, \mathbf{u}), NotCoExistence(\mathbf{u}, v)\}$

Fig. 2: The MailOfMine local static constraint diagrams

the reader can see the aforementioned constraints having it as the implied (see Figures 2a and 2c). Finally, Figure 2d suggests that, if u is performed, neither before (above) nor afterwards (below) you are allowed to do v. Anything else is admitted ($*$). This is the sense of $NotCoExistence(u, v)$, as drawn on the right of u. On the left, the constraint which u was involved in as an implied task, that is $RespondedExistence(t, u)$, is depicted (see Figure 2b).

The local view can focus on a possible sub-trace of executed tasks, as in Figure 3. The meaning of symbols is the same as before, although here the focus is moved on what could have happened (or is allowed to happen) if t is performed after u, whereas diagrams in Figure 2 consider the enactment of an only task at a time.

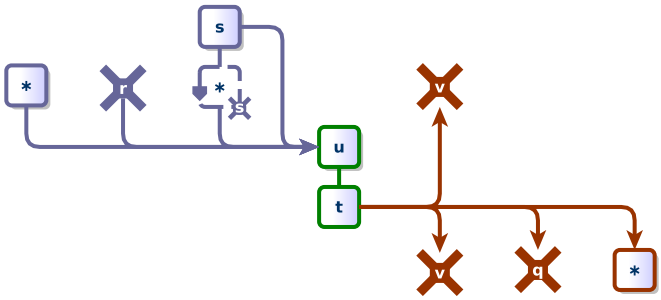
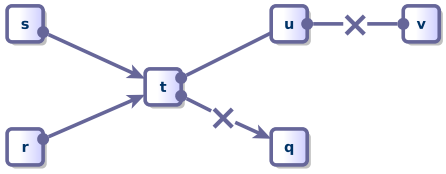


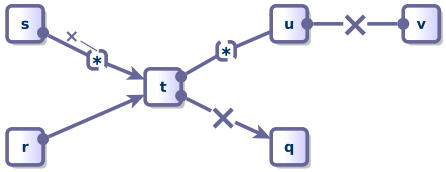
Fig. 3: The $< u, t >$ tasks subtrace constraints diagram

The global view. The aim of the global view (Figure 4) is to show the relations between tasks, namely *(i)* whether the presence of one implies a further constraint (on the graph, a dot on the tail of an arrow, starting from the implying task and ending on the implied), *(ii)* which task must be performed after, between the implying and the implied, if known (on the graph, an arrow, put on the head or the tail), *(iii)* whether the presence of one implies the absence of another (a cross in the middle of the arrow), or not (no cross put upon). All of the previous information bits are independent of each other, hence all the possible combinations are allowed. This is the restricted basic graphical syntax used in Figure 4a. Indeed, it is not explicitly expressed how strong the constraint is (e.g., whether other tasks can be performed between the implying and the implied), in order to tidy the diagram up and provide a fast view of the overall process, without entering in details that are likely better explained through the local views: they can rely, in fact, on dimensions spread on axes the cartesian way, not as in graphs.

Nonetheless, skilled users might want to have a complete vision of the constraints involved, even though it might result in a reduced readability, due to the unavoidable increase of graphical symbols to draw in the diagram. Thus, a richer graphical syntax is needed. Its design rationale is to extend the basic,



(a) Basic



(b) Extended

Fig. 4: The MailOfMine global static constraints diagram

though keeping coherence with *(i)* the visual language terms used and *(ii)* the graph structure. This allows the user to be required of a minimal cognitive effort in order to learn its semantics, on one hand, and lets her toggle between the basic and the extended view. Indeed, only arcs are loaded with new symbols, as depicted on Figure 4b: no additional shape nor any change in the graph topology are required.

Both diagrams in Figure 4 draw the same set of constraints that were locally represented by example in Figure 2.

The global view is inspired to the graphical syntax of [22], though its design focuses on the basic relations (before/after, implying/implied, existence/absence) between tasks in a binary constraint.

Coupling this diagram with the local view is useful for avoiding the misunderstanding that could arise by the usage of oriented graphs. Indeed, Finite State Automata, Petri Nets, State Transition Networks, UML Activity Diagrams, Flowcharts, and so forth, all share a common interpretation: roughly speaking, nodes are places to traverse one by one, following a path that respects the direction given by arrows along the arcs. Here, it is not the case: e.g., considering Figure 4a, one could intuitively suppose that, done s, the next task is t. It is not true: after s, r or u could be performed, even many times, and after some further passages finally t.

A GUI sketch. Figure 5 draws a prototype of the window showing a local view, on the task t. The additional information regarding the cardinality of the task, as far as the actors involved and so forth, is located on the bottom of the window. The global view, put on the right, is used as a navigation tool on the process schema. Conversely, at any point in time it will be possible to activate the local view of a task selected on the global view screen, in order to freely switch from one to another.

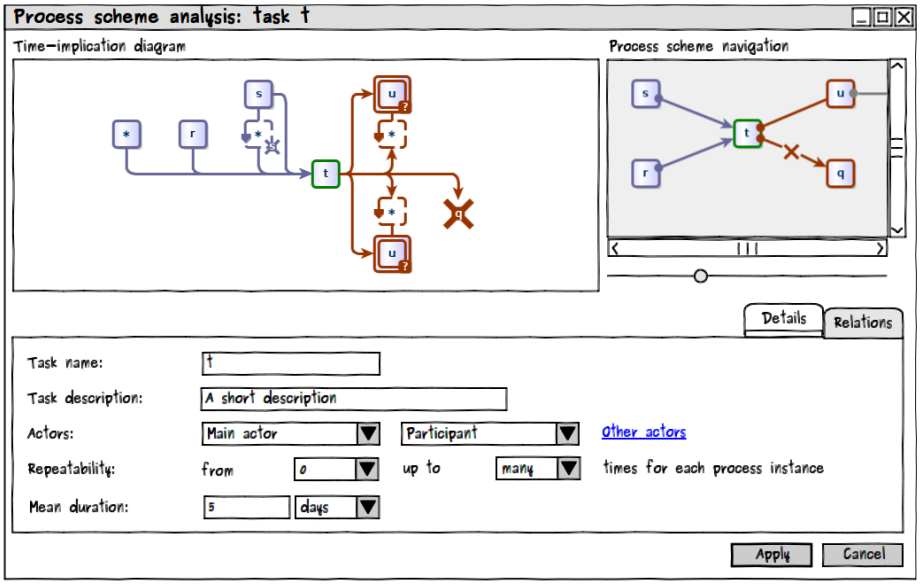


Fig. 5: The task details screen

5.2 Running Instances

A dynamic view is associated to the static process scheme, for the management of running instances. Such a view is designed to be interactive, i.e., to let the user play with the model, so to control the evolution of the running process. Moreover, she can better learn the constraints mechanism by looking at the process evolving. Indeed, it is based on the same visual notation provided for the visualization of constraints (see Figure 3), based in turn on local view diagrams. This choice is made in order to remark the user that global views do not explicitly express the evolution of the system over time, whereas local views do. Figure 6 depicts a sample evolution of a process instance.

From a starting step onwards, the user is asked to specify which the next task to perform is. At each step, the following tasks that can be enacted are shown, by means of the same visual language used for static views. After one of them is fired, all the *possible* and *mandatory* following tasks are shown. And so forth. We recall here that the recognition of the possible initial tasks, as far as the evolution which follows, is a view on the current state of the FSA obtained as the intersection of all the FSA's expressing the constraints in the process scheme. Figure 7 is a prototype sketch.

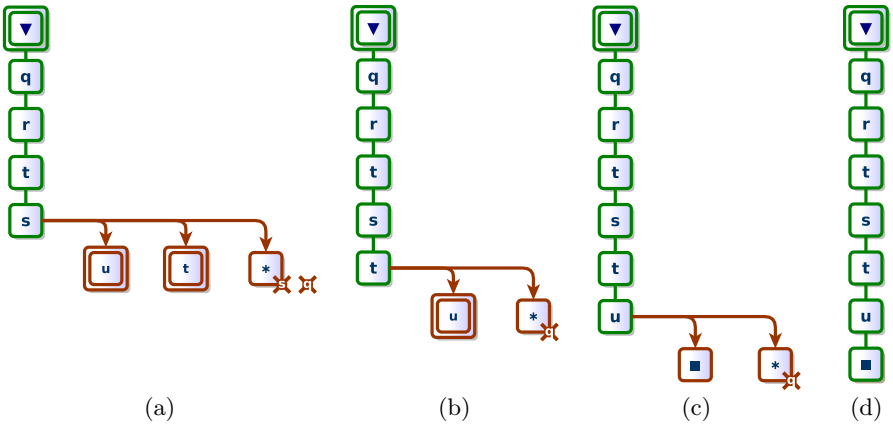


Fig. 6: The MailOfMinedynamic process view

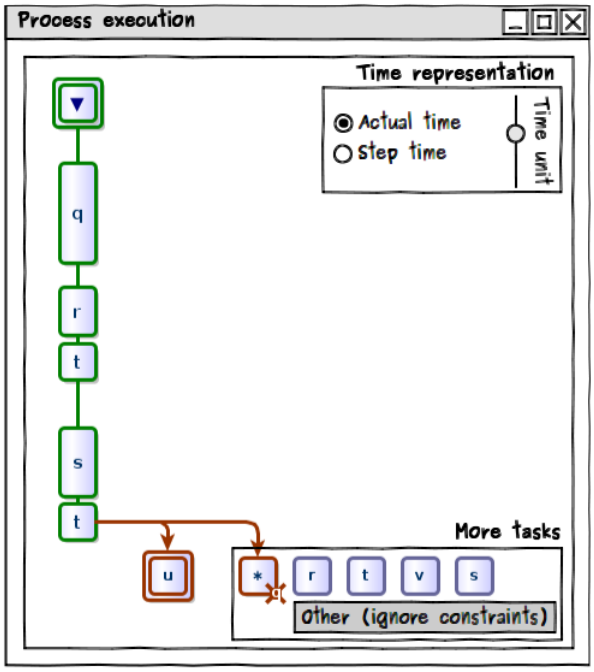


Fig. 7: The process execution management window

6 Related Work

Text Mining, or *Knowledge Discovery from Text*, deals with the machine supported analysis of text: indeed, it refers generally to the process of extracting interesting information and knowledge from unstructured text. Though text mining covers many topics that are out of the scope of this paper (see [23, 24] for comprehensive surveys), we list some techniques that are instead relevant to our work.

[25] proposes a method employing text mining techniques to analyze email messages collected at a customer center. Based on [26], this work shows how to cope with the information extraction in the context of email messages, for the construction of a key concept dictionary. Our aim is not restricted to the extraction of the key concept dictionary, but rather deals with the mining of activities performed on top of them; on the other side, in MailOfMine, we assume to rely on a user-provided dictionary of keywords, as described in Section 3.2. [14] introduces the usage of an ontology of email acts, i.e., Speech Acts [13], to classify messages.

Process Mining [27], a.k.a. *Workflow Mining* [28], is the set of techniques that allow the extraction of structured process descriptions, stemming from a set of recorded real executions. Such executions are intended to be stored in so called *event log*s, i.e., textual representations of a temporarily ordered linear sequence of tasks. There, each recorded *event* reports the execution of a *task* (i.e., a well-defined step in the workflow) in a *case* (i.e., a workflow instance). Events are always recorded sequentially, even though tasks could be executed in parallel: it is up to the algorithm to infer the actual structure of the workflow that they are traces of, identifying the causal dependencies between tasks (*condition*s). ProM [29] is one of the most used plug-in based software environment for implementing workflow mining techniques.

Most of the mainstream process mining tools model processes as Workflow Nets (WFNs – see [28]), explicitly designed to represent the control-flow dimension of a workflow. From [30] onwards, many techniques have been proposed, in order to address specific issues: pure algorithmic (e.g., α algorithm, drawn in [31] and its evolution α^{++} [32]), heuristic (e.g., [33]), genetic (e.g., [34]), etc. Indeed, heuristic and genetic algorithms have been introduced to cope with noise, which the pure algorithmic techniques were not able to manage. A very smart extension to the previous research work has been recently achieved by the two-steps algorithm proposed in [35].

EMailAnalyzer [36] is an integrated ProM plug-in for mining processes from email logs, that are XML files compatible with the ProM framework, built through the analysis of *(i)* senders and receivers, in order to guess the actors involved, and *(ii)* tags on email messages and subjects, for extracting the task. E-mail messages are extracted from an Outlook archive. Our approach shares similar ideas for the disambiguation of actors and sociograms, and aims at extending it by *(i)* building a plug-in based platform capable to retrieve email messages from multiple archives (not only Outlook), and *(ii)* extracting cases

and relations among tasks from a more comprehensive analysis of email fields (headers, threads, body, attachments, etc.).

The need for flexibility in the definition of processes leads to an alternative to the classical "imperative": the "declarative" approach. Rather than using a procedural language for expressing the allowed sequences of activities, it is based on the description of workflows through the usage of constraints: the idea is that every task can be performed, except what does not respect them. [22] shows how the declarative approach can help in obtaining a fair trade-off between flexibility in managing collaborative processes and support in controlling and assisting the enactment of workflows. Such constraints, in [37] (Chapter 6) are formulations of Linear Temporal Logic ([38] – Chapter 3). DecSerFlow [39] and ConDec [40] (now named Declare [41, 42]) provide graphical representations for processes described through the declarative approach. Nonetheless, we believe that the declaration of collaborative workflows constraints can be expressed by means of regular expressions, rather than LTL formulae: regular expressions express finite languages (i.e., processes with finite traces, where the number of enacted tasks is limited). LTL formulae are thought to be used for verifying properties over semi-infinite runs instead. On the contrary, human processes have an end, other than a starting point.

The Declare framework [41] is a concrete implementation of a constraint-based process management system, supporting multiple declarative languages. Declare Miner is the plug-in for mining declarative processes from within the ProM framework. [43] described the usage of Inductive Logic Programming Techniques to mine models expressed as a $\mathcal{S}$CIFF ([44]) theory, finally translated to the ConDec notation.

7 Conclusions

We are currently in the process of realizing the various techniques into a working prototype and then validating it over a large email messages collection.

Up to now, once analyzed in depth the current literature in order to figure out how to extend or specialize them to our purpose (Section 6), we defined a modular architecture for the realization of the approach (Section 3). We established the theoretical basis for our work, in terms of modeling entities for the expression of declarative workflows (Section 2) and graphical notations for their visualization (Section 5). We programmed the preliminary module for storing the email-related information in a database, on top of `eml` raw email messages, retrieved via IMAP or out of text mail archives. We built a fair reliable algorithm for clustering mail conversations into threads (Section 4) and a mining algorithm for discovering declarative models of processes out of traces [17], too. We are currently working on the rest of the modules, currently stub-based.

We would like to note that this work aims at addressing the open research challenge of dealing with mixed logs, i.e., logs in which traces from multiple processes are present. Most of existing workflow mining approaches suppose to treat logs in which only traces of the same process are present; conversely,

an email collection can be abstracted as a log containing traces from different processes.

Future work includes the extension of the approach to multiple users' email messages collections, and a more extensive validation. Moreover, we plan to deal with privacy concerns that naturally arise when mining over personal email messages. Indeed, it is an open issue how to perform private object matching, and the case of email messages is particularly challenging due to the already limited amount of fields that can be used to perform the matching task.

We want to apply the MailOfMine methodology on the field of collaborative activities of Open Source software development, reported by mailing lists [45].

References

1. Warren, P., Kings, N., Thurlow, I., Davies, J., Buerger, T., Simperl, E., Ruiz, C., Gomez-Perez, J.M., Ermolayev, V., Ghani, R., Tilly, M., Bösser, T., Imtiaz, A.: Improving knowledge worker productivity - the Active integrated approach. BT Technologiy Journal **26**(2) (2009) 165–176
2. Catarci, T., Dix, A., Katifori, A., Lepouras, G., Poggi, A.: Task-centred information management. In: DELOS Conference. Volume 4877 of Lecture Notes in Computer Science., Springer (2007) 197–206
3. Innocenti, P., Ross, S., Maceciuvite, E., Wilson, T., Ludwig, J., Pempe, W.: Assessing digital preservation frameworks: the approach of the SHAMAN project. In Chbeir, R., Badr, Y., Kapetanios, E., Traina, A.J.M., eds.: MEDES, ACM (2009) 412–416
4. Heutelbeck, D.: Preservation of enterprise engineering processes by social collaboration software. Personal communication (2011)
5. Smart Vortex Consortium: Smart Vortex – Management and analysis of massive data streams to support large-scale collaborative engineering projects. FP7 IP Project
6. De Giacomo, G., Mecella, M.: On the representation of constraints for declarative models. Private communication (April 2010)
7. Chomsky, N., Miller, G.A.: Finite state languages. Information and Control **1**(2) (1958) 91–112 The equivalence between regular grammars and FSAs is demonstrated here.
8. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3** (April 1959) 114–125 A comprehensive study on ASFs: it contains the demonstration of the equivalence between ASFDs and ASFNDs.
9. Gisburg, S., Rose, G.F.: Preservation of languages by transducers. Information and Control **9**(2) (1966) 153 – 176 Many closure properties about regular languages are demonstrated here.
10. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM, IEEE (2011) 192–199
11. de Carvalho, V.R., Cohen, W.W.: Learning to extract signature and reply lines from email. In: CEAS. (2004)
12. Myers, E.W.: An O(ND) difference algorithm and its variations. Algorithmica **1**(2) (1986) 251–266
13. Searle, J. In: A Taxonomy of Illocutionary Acts. University of Minnesota Press, Minneapolis (1975) 334–369

14. Cohen, W.W., Carvalho, V.R., Mitchell, T.M.: Learning to classify email into "speech acts". In: EMNLP, ACL (2004) 309–316
15. Weigand, H., de Moor, A.: Workflow analysis with communication norms. Data Knowl. Eng. **47**(3) (2003) 349–369
16. Di Ciccio, C., Mecella, M.: Mining constraints for artful processes. In W. Abramowicz, D. Kriksciuniene, V.S., ed.: 15th International Conference on Business Information Systems. Volume 117 of Lecture Notes in Business Information Processing., Springer (2012) (to appear).
17. Di Ciccio, C., Mecella, M.: MINERful, a mining algorithm for declarative process constraints in MailOfMine. Technical report, Dipartimento di Ingegneria Informatica, Automatica e Gestionale "Antonio Ruberti" – SAPIENZA, Università di Roma (2012)
18. Zardetto, D., Scannapieco, M., Catarci, T.: Effective automated object matching. In Li, F., Moro, M.M., Ghandeharizadeh, S., Haritsa, J.R., Weikum, G., Carey, M.J., Casati, F., Chang, E.Y., Manolescu, I., Mehrotra, S., Dayal, U., Tsotras, V.J., eds.: ICDE, IEEE (2010) 757–768
19. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering **19** (2007) 1–16
20. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval. ACM Press / Addison-Wesley (1999)
21. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley (1989)
22. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D **23**(2) (2009) 99–113
23. Hotho, A., Nürnberger, A., Paaß, G.: A brief survey of text mining. LDV Forum - GLDV Journal for Computational Linguistics and Language Technology **20**(1) (May 2005) 19–62
24. Berry, M.W., Castellanos, M., eds.: Survey of Text Mining II: Clustering, Classification, and Retrieval. Springer (September 2007)
25. Sakurai, S., Suyama, A.: An e-mail analysis method based on text mining techniques. Appl. Soft Comput. **6**(1) (2005) 62–71
26. Sakurai, S., Ichimura, Y., Suyama, A., Orihara, R.: Acquisition of a knowledge dictionary for a text mining system using an inductive learning method. In: Proceedings of IJCAI 2001 Workshop on Text Learning: Beyond Supervision. (2001) 45–52
27. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
28. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1) (1998) 21–66
29. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, E., Weijters, T.: Prom: The process mining toolkit. In de Medeiros, A.K.A., Weber, B., eds.: BPM (Demos). Volume 489 of CEUR Workshop Proceedings., CEUR-WS.org (2009)
30. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In Schek, H.J., Alonso, G., Saltor, F., Ramos, I., eds.: Advances in Database Technology EDBT'98. Volume 1377 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1998) 467–483 10.1007/BFb0101003.
31. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004) 1128–1142

32. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. **15**(2) (2007) 145–180
33. Weijters, A., van der Aalst, W.: Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering **10** (2001) 2003
34. Medeiros, A.K., Weijters, A.J., Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Discov. **14**(2) (2007) 245–304
35. van der Aalst, W., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., Gnther, C.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling **9** (2010) 87–111 10.1007/s10270-008-0106-z.
36. van der Aalst, W.M.P., Nikolov, A.: Mining e-mail messages: Uncovering interaction patterns and processes using e-mail logs. IJIIT **4**(3) (2008) 27–45
37. ter Hofstede, A.M., van der Aalst, W.M.P., Adamns, M., Russell, N., eds.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
38. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (2001)
39. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In Bravetti, M., Núñez, M., Zavattaro, G., eds.: WS-FM. Volume 4184 of Lecture Notes in Computer Science., Springer (2006) 1–23
40. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In Eder, J., Dustdar, S., eds.: Business Process Management Workshops. Volume 4103 of Lecture Notes in Computer Science., Springer (2006) 169–180
41. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: EDOC, IEEE Computer Society (2007) 287–300
42. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In Meersman, R., Tari, Z., eds.: OTM Conferences (1). Volume 4803 of Lecture Notes in Computer Science., Springer (2007) 77–94
43. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. T. Petri Nets and Other Models of Concurrency **2** (2009) 278–295
44. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The sciff framework. ACM Trans. Comput. Log. **9**(4) (2008)
45. Morera-Mesa, A., Di Ciccio, C.: On the mining of processes out of open source development mailing lists. Private communication (June 2011)