Representing and Visualizing Mined Artful Processes in MailOfMine*

Claudio Di Ciccio, Massimo Mecella, and Tiziana Catarci

Sapienza – Università di Roma
Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti
{cdc|mecella|catarci}@dis.uniroma1.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". MailOfMine is a tool, the aim of which is to automatically build, on top of a collection of *e*-mail messages, a set of workflow models that represent the artful processes laying behind the knowledge workers activities. This paper presents its innovative graphical syntax proposal and the interface for representing and showing such mined processes to users.

Keywords: process mining, process visualization, artful process

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" [18]. 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 not exactly reproducible even by their originators – since they are not written down – and can not be easily shared either. Their outcomes and their information exchanges are done very often by means of *e*-mail 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:

* 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. The authors would like also to thank Monica Scannapieco and Diego Zardetto for useful insights and discussions.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-25364-5_9

- *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) [9,18];
- *information warfare*, especially in supporting anti-crime intelligence agencies: let us suppose that a government bureau is able to access the *e*-mail 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. Though, a system should 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 inferring the criminal activities of the suspected person(s);
- *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 [1,2,14].

The objective of the MailOfMine approach, proposed in [12], which this paper is based upon, is to automatically build, on top of a collection of *e*-mail messages, a set of workflow models that represent the artful processes laying behind the knowledge workers activities. Here, we discuss how we addressed the challenge of showing artful processes to users, i.e., knowledge workers, 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 symbols that describe processes.

The work here presented is related to the so called *process mining*, a.k.a. *workflow mining* [4], that is the set of techniques allowing the extraction of structured process descriptions, stemmed from a set of recorded real executions (stored in the *event log*s). ProM [5] 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 [3]), explicitly designed to represent the control-flow dimension of a workflow, and visualizes processes in various graphical notations, all of them basically graph-based. Also, the most of standard graphical notations for processes, such as UML Activity Diagrams and BPMN – Business Process Modeling Notation, basically visualize an entire process as a graph, in order to provide the users with a one-shot view of the process.

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

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9

is that every task can be performed, except what does not respect them. Such constraints, in [15] (Chapter 6) are formulations of Linear Temporal Logic ([11] – Chapter 3). DecSerFlow [6] and ConDec [16] 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. We envision the process schemes like grammars describing the language spoken by collaborative organisms in terms of activities, thus being more related to formal languages rather than temporal logic. In this paper we propose a novel graphical notation for such languages.

2 The MailOfMine Approach

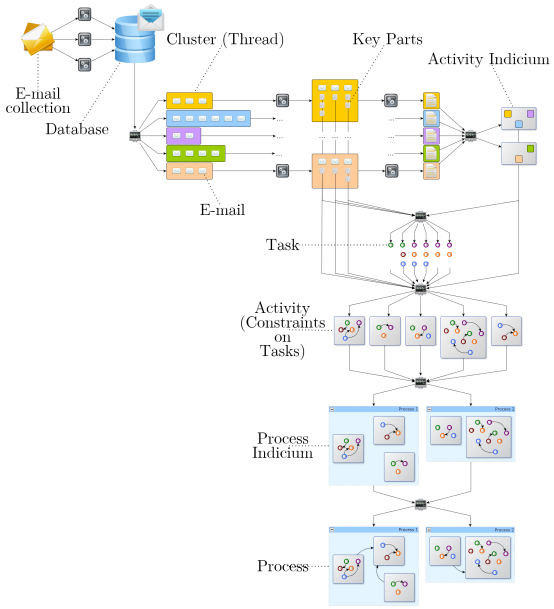


Fig. 1: The MailOfMine approach [12]

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. The architecture is shown in Figure 1; before briefly presenting it, we need to introduce some basic concepts needed in the following. The details of the architecture, as well as the formal definitions of concepts, are presented in [12].

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-25364-5_9

An *actor* is the subject directly or indirectly taking part in the progress of a work. A *task* is an elementary unit of work. Each task is connected to *(i)* its expected *duration*, *(ii)* zero or more *outcome*s, *(iii)* one or more *actor*s. An *activity* is a collection of tasks or (recursively) other activities. A *key part* is each *unique* piece of text belonging to the *e*-mail messages exchanged in a communication trace. Thus, given a collection of duplicated pieces of text (coming from the same or different *e*-mail messages in the trace), just a single representative is selected as key part. For instance, HTML signatures used by the sender of the *e*-mail, quotations of previous *e*-mail messages used in replies, etc., are all examples of redundant information that may appear in a communication trace, but is filtered out by means of the *key part* concept. On the other hand, any piece of text not appearing in any other *e*-mail message in the discussion thread is interpreted as a *key part*. An *indicium* is any communication trace, or part of it, attesting the execution of a task, an activity, or a process instance.

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. Constraints do not force the process instance to follow a tight sequence, but rather leave it the flexibility to follow different paths, while performing the execution, though respecting a set of rules that avoid illegal or non-consistent states.

In [12], we argued that each constraint is expressible through regular grammars. Regular grammars are recognizable through Finite State Automata (FSA) [10] (either deterministic or non-deterministic [17]). The FSA recognizing the correct traces for processes (i.e., accepting the valid strings) is the intersection of all the FSAs composing the set of constraints. This is the theoretical basis for the reasoning core that internally manages the process scheme to be finally shown to the user.

Initially, we need to extract *e*-mail messages from the given archive(s). 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. Once the communication threads are recognized, we can assume them all as activity indicia candidates. Afterwards, messages are analyzed in order to identify key parts. Key parts are gathered by combining the technique for the removal of quoted material, presented in [8], with an iterative approach over the *e*-mail messages in the thread. MailOfMine can thus build the activity indicia as the concatenation of all the key parts belonging to the messages of a communication thread. Then, the clustering algorithm is used again, this time to identify the matches between activity indicia. By taking into account the activities set and the key parts (task indicia candidates), the clustering algorithm checks for matching key parts: those are considered tasks.

Taking as input all of the preceding outcomes, MailOfMine starts searching for execution constraints between tasks within the activities they belong to. To this intent, we exploit and extend the SPIRIT techniques for regular patterns mining ([13]). Specifically, on the basis of all the possible constraints, a selection of those that are valid over most of the activity indicia is made.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9

Once activities and tasks are recognized, a supervised learning process takes place, in order to cluster activities into processes. Once processes are identified, MailOfMine performs the second step for the construction of the process scheme, i.e., the mining of production rules among activities inside the same process, by using the SPIRIT techniques for regular pattern mining again.

3 Process Visualization

The literature dealing with the representation of processes typically aims at visualizing the processes all at once, by means of diagrams that show the complete grid of interconnections among activities. Here we propose a change in the viewpoint. As stated before, 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 order on activities, neither explicit nor implicit. For instance, it is not mandatory to specify which the initial activity is, and the following step depends on the previous choices. 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 3.1) and *(ii)* an interactive evolutionary graphical representation for the visualization of running instances (dynamic view: Section 3.2). 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.

3.1 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., t) is either *(i)* directly implied (e.g., if s is done, then t must be done), or *(ii)* directly implying (e.g., if t is done, no matter when, u 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 rationale of having a local view only. For

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-25364-5_9

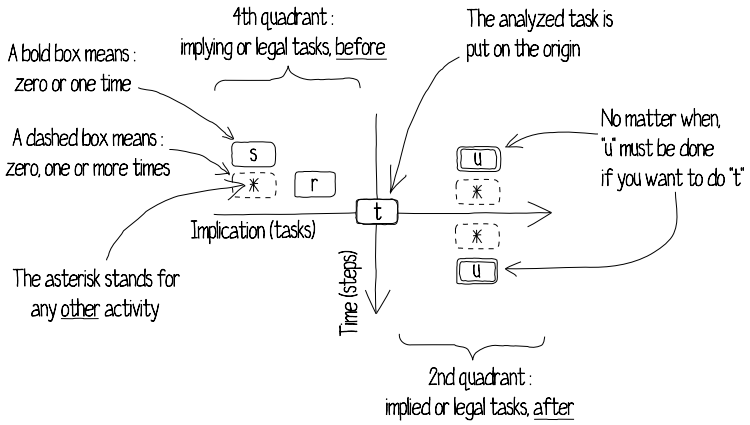


Fig. 2: The tasks static view rationale

instance, if r was a task implying that s could not be done further (and if s is done, then t must be done), such a chain of constraints is not taken into account, unless we are looking at the constraints regarding t. This avoids the confusion of too many cross-implications to consider at a time (for sake of readability) and respects the principle of declarative approaches: you can not do s if r was done; though, if r was not done, no constraint on s must hold, then we do not care it as a side-effect on t.

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 implication is a binary set (implying, implied). The repeatability is a space of 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.

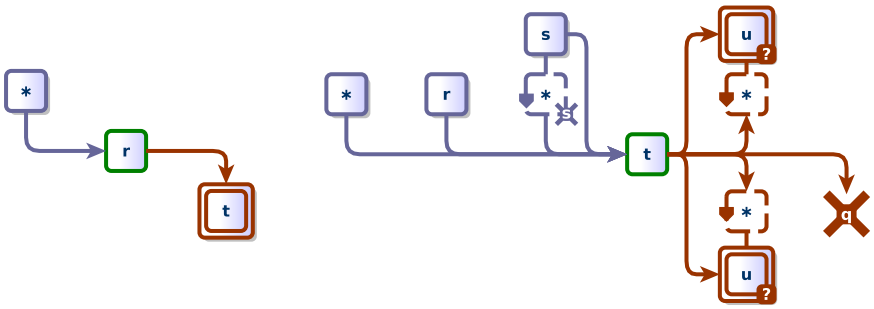
Our graphical notation represents time and implication as the coordinates of a bidimensional drawing, where time is on the ordinates. This ideal y axis divides the plane space into two separate regions: one for each value of the implication dimension (implying, implied), on the abscissae. The x axis 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. Indeed, on the origin of this chart, inspired to the cartesian coordinate system, we put the task under examination. See Figure 2 for a sketch of the rationale.

Differently from the classical orientation, we consider the y axis oriented towards the bottom. This for following the reading directionality. For the same reason, the implication relation order flows from the left to the right. Of course, this detail 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.

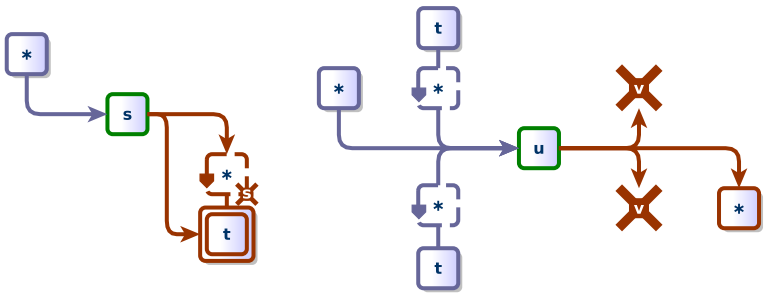
Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9

The repeatability is expressed by the thickness of the boundaries round the boxes representing tasks: dashed for tasks that can be done zero, one or more times, bold 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 t 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 of the single task in 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 which are interested in focused constraints. 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 s can not be executed before t, a $*$ before t means "any task, except s). Examples of the diagrams are in Figure 3.



(a) $\{chainSuccession(r,t)\}$ (b) $\{respondedExistence(t,u), negationResponse(t,q)\}$



(c) $\{alternateResponse(s,t)\}$ (d) $\{respondedAbsence(u,v)\}$

Fig. 3: The MailOfMine local static constraint diagrams

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-25364-5_9

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 overall rationale is the same: such diagrams must be easy to be sketched by a pen, as well.

The local view can focus on a possible sub-trace of executed tasks, as in Figure 4.

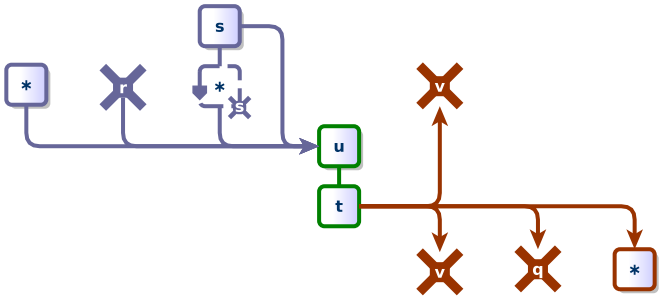
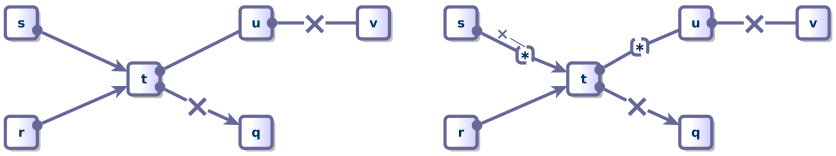


Fig. 4: The $< u, t >$ tasks subtrace constraints diagram

The global view. The aim of the global view (Figure 5) 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 5a. 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, 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

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9



(a) Basic (b) Extended

Fig. 5: The MailOfMine global static constraints diagram

depicted on Figure 5b: no additional shape nor any change in the graph topology are required.

The global view is inspired to the graphical syntax of [7]; only a minimal subset of the DecSerFlow constraints are represented: this makes it easier for future developers to introduce new constraints, that inherit the basic relations (before/after, implying/implied, existence/absence), without introducing new graphical notations, but only modifying the local view graphical patterns.

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 use the same semantics: 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 5, one could intuitively suppose that, done t, the next task is u. It is not true: after t, s or t itself could be performed, even many times, and only after a while, u.

A GUI sketch. Figure 6 draws a prototype of the window showing a local view, on the t task. The additional information regarding the cardinality of the task, 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, it will be possible at any point in time to activate the local view of a task selected on the global view screen, in order to freely switch from one to another.

3.2 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 traces constraints visualization (see Figure 4), 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 7 depicts a sample evolution of a process instance.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9

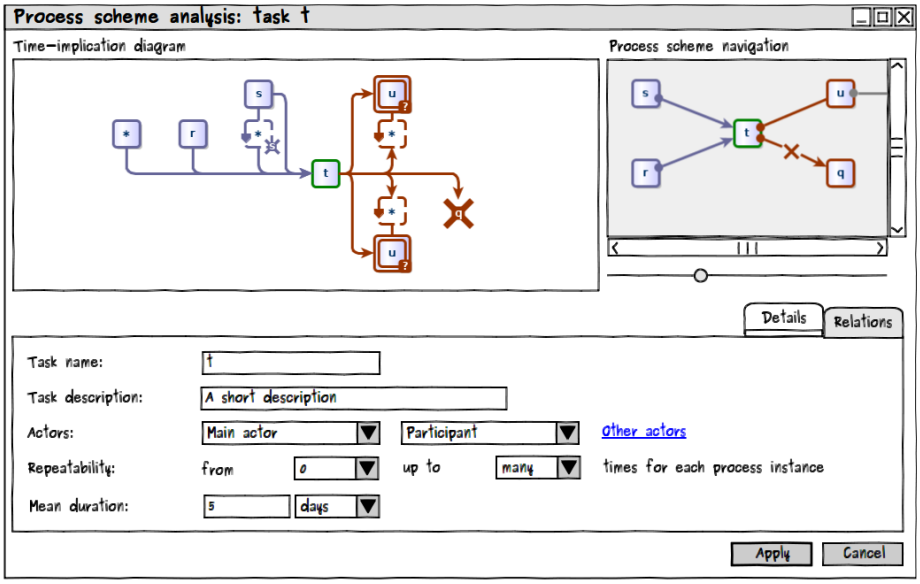


Fig. 6: The task details screen

From a starting task 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. We are currently implementing such process scheme viewer.

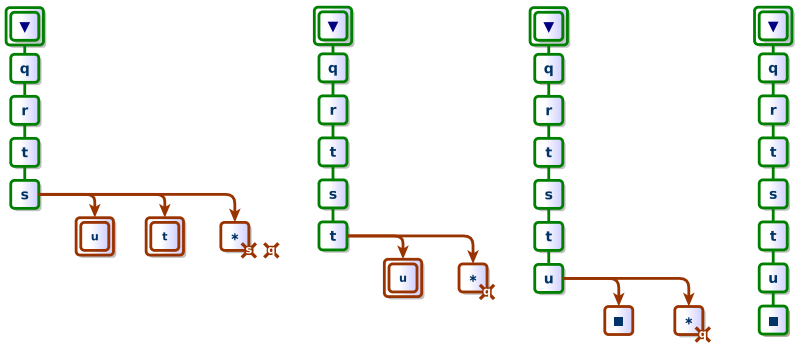


Fig. 7: The MailOfMine dynamic process view

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9

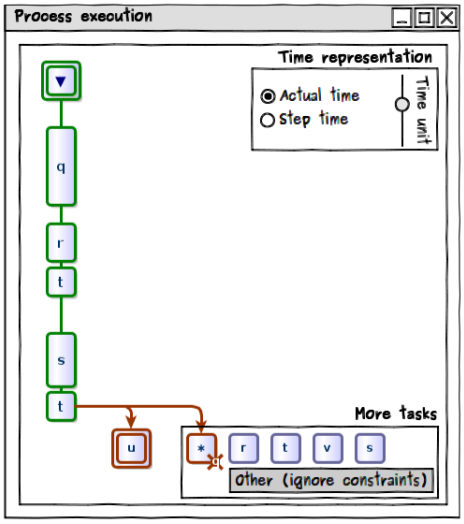


Fig. 8: The process execution management window

A GUI prototype sketch. Figure 8 is a prototype sketch. It remarks two main features. The first is that users can adapt the timing in two different ways: either *(i)* as if every task lasts a time unit only, ignoring pauses between the preceding and the following, or *(ii)* showing the actual time consumption for both tasks completion and pauses in between. The former is useful for a compact representation, the latter for a realistic snapshot of the time the running process is taking, with the evidence of delays. The second remarked feature is that users may even violate constraints: artful processes are subject to frequent changes, thus imposing a strict respect of constraints could be frustrating for the user who would like to do something else. This, on the other hand, can be a useful information for the process miner, since it can in turn refine the evolution of the process scheme itself, if a sufficient quantity of deviations from the expected paths are detected. For the next tasks to take over, the user will be asked to choose whether she wants to *(i)* delete the violated constraint from the overall process scheme, or *(ii)* proceed as if it was a point deviation only, namely keeping the constraints untouched. The former option is manageable thanks to the fact that each constraint is an FSA: this allows to immediately identify the violated constraint, on one hand, and recalculate the updated process scheme, on the other hand. During the execution of the process instance, in fact, not only the validity of the path on the global intersection FSA is considered: every step is monitored by the evolution of the individual constraint FSA components as well. So, when one or more of them are violated, they can be deleted from the set, on top of which the intersection FSA is computed. Once removed, the FSA is recalculated. Finally, the same history, up to the deviation, is enacted back on the new FSA. The next possible tasks to perform are shown accordingly.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9

4 Conclusions

In this paper we outlined the MailOfMine approach for mining artful processes from *e*-mail messages collections, focusing on the visualization aspects, i.e., the graphical syntax, the diagrams drawn and its GUI. We are currently in the process of realizing the various techniques into a working prototype. Then, we are going to validate it over a large *e*-mail messages collection. At the same time, we are testing the validity of our user interface with knowledge workers. The idea is to propose a graphical language for expressing artful processes, tailored to the users who will actually interact with it the most. Thus, symbols, connectors and all the other graphical details for modeling processes will be validated with them, rather then decided a priori by a team of experts in the business process management domain. In other words, we will exploit a user centered design for developing not the application only, but its core graphical language too.

References

1. Shaman. FP7 IP Project: `http://shaman-ip.eu/shaman/`
2. Smart Vortex. FP7 IP Project: `http://www.smartvortex.eu/`
3. van der Aalst, W.M.P.: Verification of workflow nets. ICATPN, LNCS 1248 (1997)
4. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
5. 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 BPM 2009 Demos
6. van der Aalst, W.M.P., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In WS-FM. LNCS 4184 (2006)
7. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D 23(2) (2009)
8. de Carvalho, V.R., Cohen, W.W.: Learning to extract signature and reply lines from email. In CEAS (2004)
9. Catarci, T., Dix, A., Katifori, A., Lepouras, G., Poggi, A.: Task-centred information management. In DELOS Conference. LNCS 4877 (2007)
10. Chomsky, N., Miller, G.A.: Finite state languages. Information and Control 1(2), 91–112 (1958)
11. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (2001)
12. Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: MailOfMine – Analyzing mail messages for mining artful collaborative processes. In SIMPDA (2011)
13. Garofalakis, M.N., Rastogi, R., Shim, K.: Spirit: Sequential pattern mining with regular expression constraints. In VLDB 1999
14. Heutelbeck, D.: Preservation of enterprise engineering processes by social collaboration software (2011), personal communication
15. ter Hofstede, A.M., van der Aalst, W.M.P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
16. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In BPM Workshops. LNCS 4103 (2006)
17. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3, 114–125 (April 1959)
18. Warren, P., Kings, N. et al: Improving knowledge worker productivity - The ACTIVE integrated approach. BT Technology Journal 26(2), 165–176 (2009)

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-642-25364-5_9

This document is a pre-print copy of the manuscript
(Di Ciccio, Catarci, and Mecella 2011)
published by Springer (available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-642-25364-5_9

References

Di Ciccio, Claudio, Tiziana Catarci, and Massimo Mecella (2011). "Representing and Visualizing Mined Artful Processes in MailOfMine". In: *HCI-KDD*. Ed. by Andreas Holzinger and Klaus-Martin Simonic. Vol. 7058. Lecture Notes in Computer Science. Springer, pp. 83–94. ISBN: 978-3-642-25363-8. DOI: 10.1007/978-3-642-25364-5_9.

BibTeX

```
@InProceedings{   DiCiccio.etal/HCI-KDD2011:RepresentingVisualizingDeclarativeModels,
  author        = {Di Ciccio, Claudio and Catarci, Tiziana and Mecella,
                   Massimo},
  title         = {Representing and Visualizing Mined Artful Processes in
                   {MailOfMine}},
  booktitle     = {HCI-KDD},
  year          = {2011},
  pages         = {83-94},
  publisher     = {Springer},
  crossref      = {HCI-KDD_USAB2011},
  doi           = {10.1007/978-3-642-25364-5_9},
  keywords      = {process mining, process visualization, artful process}
}
@Proceedings{     HCI-KDD_USAB2011,
  title         = {Information Quality in e-Health - 7th Conference of the
                   Workgroup Human-Computer Interaction and Usability
                   Engineering of the Austrian Computer Society, {USAB} 2011,
                   Graz, Austria, November 25-26, 2011. Proceedings},
  year          = {2011},
  editor        = {Andreas Holzinger and Klaus{-}Martin Simonic},
  publisher     = {Springer},
  series        = {Lecture Notes in Computer Science},
  volume        = {7058},
  doi           = {10.1007/978-3-642-25364-5},
  isbn          = {978-3-642-25363-8}
}
```