Distributed Discovery of Multi-Level Approximate Process Patterns

Antonio Turi, Annalisa Appice, Michelangelo Ceci, and Donato Malerba

Dipartimento di Informatica, Università degli Studi di Bari
via Orabona, 4 - 70126 Bari - Italy
{turi,appice,ceci,malerba}@di.uniba.it

Abstract. Process mining focuses on the discovery of knowledge about a (business) process from a set of its executions stored in an event log. Each event describes an activity and its performer. Process mining techniques allows automatically extracting the process model that gains insight into various perspectives, such as the control flow perspective, data, and organizational perspective. In this paper, we do not assume the existence of a single process model to which all process executions comply, and the goal is to discover a set of process patterns frequently occurring in the log. These patterns reveal the order of execution of events in a process, and, at the same time, they may describe properties of activities and performers participating to the events. Since activities and performers are often grouped in hierarchical categories, process patterns are discovered at multiple level of granularity. To deal with huge amount of data stored in a massive log, we resort to a distributed approximate discovery of multi-level frequent process patterns. Distinct process executions are distributed on several nodes of a Grid platform. Locally frequent patterns are discovered on each node. Global pattern set are then approximated and frequency is estimated on the basis of the local frequency of the pattern discovered at several nodes. Experiments are performed on a real event log.

1 Introduction

Process mining concerns the automated discovery of (business) process models from a set of real executions (or cases) registered in the event log of an enterprise informative system. Each event describes an activity executed by a performer within one case. The event is stored with a timestamp that records the time of occurrence and implicitly defines a total order over events. Process mining methods input events and extract knowledge (i.e., process patterns) that provides insight into the execution and organization of the recorded process [13]. These process patterns enable organizations to compare the behavior in the event log with the business conduct it would expect from its employees and other stakeholders.

Discovering process patterns requires sophisticated data mining techniques in order to face with difficulties deriving from the fact that a log describes objects which may belong to different data types (cases, activities and performers).

Separate data types are naturally modeled as several relational data tables (one for each data type), while relations between objects are expressed by foreign key constraints. This leads distinguishing between the reference objects of analysis (cases) and other task-relevant objects (activities and performers), and to represent their interactions. Several traditional data mining methods do not make this distinction, nor do they allow the representation of any kind of interaction. A further degree of complexity derives from the temporal relation between events that is implicit in the timestamp. Temporal autocorrelation requires that the effect of a property at any event may not be limited to the specific event.

Currently, many algorithms [3, 1, 14, 5] have dealt with these challenges and some of them are integrated into the ProM framework [15]. Although these methods satisfy requirements of process mining, they ignore the fact that activities and performers can be organized in hierarchies of categories (e.g., the performer of operations on a text file can be a writer or a reader). Indeed, by descending or ascending through a hierarchy, it is possible to view the same object at different levels of abstraction (or granularity). This corresponds to investigate global and local effect of underlying phenomenon. In general, the process miner can be forced to repeat independent experiments on different representations, but, in this case, the process patterns obtained for high granularity levels are not used to control search at low granularity levels (or vice versa). A further limitation is that process mining methods do not use the inferential mechanisms defined in a reasoning theory [10]. Conversely, domain independent knowledge can be used to express sequentiality between activities and simply arrive at valid conclusions regarding the (in)direct activity successor in a log process.

The system SPADA [8] offers a sufficiently complete solution to all the challenges posed by the process mining tasks in descriptive case. SPADA is able to jointly analyze properties and relations of reference objects and task relevant objects as well as rules expressing some independent knowledge. The output is a set of multi-level frequent patterns which may involve properties and relations of these several objects (relational patterns). By modeling a temporal relation (e.g., before) between events in a case, temporal autocorrelation is taken into account. Finally, hierarchies of task-relevant objects are mapped into different levels of granularity thus allowing a multi-level pattern generation.

Although these characteristics make SPADA a valid means of process mining, this solution is not feasible in practice. Indeed, frequent pattern discovery is a very complex task, particularly when data are massive log. In this paper, we present a method, namely G-SPADA, that preserves all nice characteristics of SPADA and solves computational drawbacks by *distributing* discovery of multi-level approximate process patterns on a Grid platform. Scalable parallel data mining for frequent pattern and association rule discovery has received some attention in literature [6, 16, 12]. Anyway they do not face the case data are distinguished in reference objects and task-relevant objects. To distribute process pattern discovery poses several issues. Firstly, we need a partitioning procedure to divide log data in several data partitions (or splits). Partitioning cannot ignore the relational structure of data, hence, a data split includes a subset of the

reference objects and all the task-relevant objects related to. Secondly, we need a computational framework to exploit the power of distributed computation and storage resources across the Internet. Finally, we define a schema to combine sets of local patterns and approximate global ones. This schema allows deriving an estimate of global support.

The paper is organized as follows. In the next Section we present background on multi-level frequent relational pattern discovery. Issues and solutions of the distributed discovery of approximate process patterns performed by G-SPADA are discussed in Section 3. Multi-level process patterns discovered from a real event log are commented in Section 4 and conclusions are drawn.

2 Multi-level Frequent Relational Pattern Discovery

The multi-level relational pattern discovery task is formally defined as follows:
Given

- a set S of reference objects,
- some sets R_k, $1 \leq k \leq m$ of task-relevant objects,
- a background knowledge BK which includes hierarchies H_k on the objects in R_k and domain knowledge in form of rules,
- a deductive database D that is formed by an extensional (D_E) part where properties and relations of reference objects and task-relevant objects are expressed in derived ground predicates and an intensional part (D_I) where domain knowledge in BK is expressed in form of rules,
- M granularity levels in the descriptions (1 is the highest, M is the lowest),
- a set of granularity ψ_k which associate each object in H_k with a granularity level to deal with several hierarchies at once,
- a threshold $minsup[l]$ for each granularity level l ($1 \leq l \leq M$),
- a language bias LB to constrain the search space;

Find, for each granularity level l, the *frequent* relational patterns which involve properties and relations of task relevant-objects at level l of H_k .

In this relational setting [4], a unit of analysis (or example) $D[s]$ includes a reference object $s \in S$ and all the task relevant objects of R_k which are (directly or indirectly) related to s according to some foreign key path in D. The frequency (support) of a pattern is based on the number of units of analysis, i.e., reference objects, that the rule covers.

Example 1. Let D_E be the extensional part of a deductive database D where an information system stores data (activity and its performer) on events of a business process, in particular, the type of the activity and the timestamp. An instance of D_E is in the form:
case(c1). case(c2). ...
activity(c1,a1). activity(c1,a2). activity(c1,a3). activity(c2,a4). ...
time(a1,10). time(a2,25). time(a3,29). time(a4,13). ...

user(a1,u1). user(a2,u1). ... description(a1,create). ...

The reference objects are the set of cases (c1, c2), while the task relevant objects are the collections of performers (u1, u2) and activities (a1, a2, a3, a4). The unit of analysis D[c1] is defined as follows:

case(c1). activity(c1,a1). activity(c1,a2). activity(c1,a3).
time(a1,10), time(a2,25). user(a1,u1). user(a2,u1). description(a1,create).

Definition 1. *A relational pattern is in the form P [s]. P is a set of predicates (or items), that is,* $p_0(t_0^1), p_1(t_1^1, t_1^2), p_2(t_2^1, t_2^2), \ldots, p_m(t_m^1, t_m^2)$*, with: (i)* $p_0(t_0^1)$ *is the key predicate* [1] *over D, and (ii)* $\forall i = 1, \ldots, m$*,* $p_i(t_i^1, t_i^2)$ *is either a structural predicate* [2] *or a property predicate*[3] *or an is_a predicate* [4] *over D. s is the support that estimates the probability* $p(P)$ *on D, that is, s% of the units of analysis in D is covered by P.*

By assigning P with an existentially quantified conjunctive formula $eqc(P)$ obtained by turning P into a Datalog query, we can provide a formal definition of support of P on D. Formally,

Definition 2. *A pattern P covers a unit of analysis* $D[s]$ *if* $eqc(P)$ *is true in* $D[s] \cup BK$*, where* $BK = D_I$*.*

Definition 3. *Let* $D(S)$ *be the set of units of analysis in* D_E *and OP denote the subset of D containing the units of analysis covered by the pattern P. The support of P is defined as* $sup(P) = |OP|/|D(S)|$*.*

An example of relational pattern is reported in Example 2.

Example 2. Let D_E be the extensional database described in Example 1. A candidate relational pattern P1 on D is in the form:

P1: *case(A), activity(A,B), is_a(B,activity), before(B,C), is_a(C,activity), description(C,workinprogress), user(B, D), is_a(D, performer)* [*72.25%*]

P1 describes the execution order of two sequential activities, namely B and C. B is executed by a generic performer. The support is 72.5%, that is, 72.5% of cases covers the pattern P.

By taking into account hierarchies on task-relevant objects, relational patterns can be discovered at multiple level of granularity.

---

[1] The key predicate $p(s)$ is a unary predicate representing the reference objects in S. the name p denotes the set S, while the term s is a variable that represents the primary key of the reference objects.

[2] A structural predicate is a binary predicate $p(t, s)$ associated with a pair of reference or task relevant objects related by a foreign key FK in D. The name p denotes FK, while the term t (s) is a variable that represents the primary key of objects.

[3] A property predicate is a binary predicate $p(t, s)$ associated with the attribute ATT of a reference or task-relevant object. The name p denotes the attribute ATT, the term t is a variable representing the primary key of the object and s is a constant which represents a value belonging to the range of ATT.

[4] An is_a predicate assigns a task-relevant object in R_k with a node at level l of its hierarchy H_k.

Example 3. Let us consider two level hierarchies defined on performers and activities defined in the followings:

administrator, user $\rightarrow$ *performer*; *namemaker, delete, workflow* $\rightarrow$ *activity*

P2 is a finer-grained relational pattern than P1 obtained by descending one level in hierarchies. P2 is in the form:

P2: *case(A), activity(A,B), is_a(B,namemaker), before(B,C), is_a(C,workflow), description(C,workinprogress), is_a(D, administrator)* [*62.5%*]

P2 provides better insight than P1 on the nature of B, C and D.

Obviously, the frequency of a pattern depends on the granularity level of task-relevant objects. The formal definition of relational pattern frequent at level l is reported below.

Definition 4. *A pattern* P *[s%] at level l is frequent if* $s \geq minsup[l]$ *and all ancestors of P with respect to* H_k *are frequent at their corresponding levels. The support s estimates the probability* $p(P)$.

In SPADA [8], multi-level relational frequent patterns are discovered according to the *levelwise* method described in [9], that is, a breadth-first search in the lattice of patterns spanned by a generality order between patterns.

Definition 5. *Given two patterns* P_1 *and* P_2, $P_1 \succeq P_2$ *denotes that* P_1 *is more general than* P_2 *or equivalently that* P_2 *is more specific than* P_1.

By adopting θ-subsumption[11] to induce a generality order over pattern spaces, the following definition can be given:

Definition 6. P_1 *is more general than* P_2 *under* θ*-subsumption, denoted as* $P_1 \succeq_\theta P_2$, *if and only if* P_1 θ*-subsumes* P_2, *that is, a substitution* θ *exists such that* $P_1\theta \subseteq P_2$.

Example 4. Let us consider the relational patterns:

$P_1 \equiv$ *is_a(B,namemaker)*

$P_2 \equiv$ *is_a(B,namemaker)* $\wedge$ *before(B,C)*

$P_3 \equiv$ *is_a(B,namemaker)* $\wedge$ *before(B,C)* $\wedge$ *is_a(C,workflow)*

whose variables are implicitly existentially quantified. Then P_1 θ-subsumes P_2 ($P_1 \succeq_\theta P_2$) and P_2 θ-subsumes P_3 ($P_2 \succeq_\theta P_3$) with substitutions $\theta_2 = \oslash$ and $\theta_1 = \{Y \leftarrow R\}$, respectively.

We say that P_i is a sub-pattern of P_j if $P_i\theta-$subsumesP_j. Patterns ordered by θ-subsumption preserves the anti-monotone property of frequent patterns, that is, the frequency of a super-pattern is less than or equal to the frequency of a sub-pattern [2]. Thanks to this anti-monotone property, patterns at a given granularity level can be explored from the most general to the most specific according to an algorithm Apriori-like [2].

By descending/ascending through a hierarchy it is possible to view the same task-relevant object at different levels of abstraction (or granularity). Relational patterns involving the most abstract task-relevant objects can be well supported.

Therefore, multi-relational data mining methods should be able to explore the search space at different granularity levels in order to find the most interesting patterns (e.g., the most supported). In the case of granularity levels defined by a containment relationship, this corresponds to explore both global and local aspects of the underlying phenomenon. Very few data mining techniques do automatically support this multiple-level analysis. The user is forced to repeat independent experiments on different representations, and results obtained for high granularity levels are not used to control search at low granularity levels (or viceversa). SPADA is a noticeable exception, since it prevents the generation of some infrequent patterns by associating each candidate pattern with backward pointers to parent patterns at higher granularity levels (inter-space parenthood).

3 Distributing Discovery of Approximate Patterns

G-SPADA boils down an event log in the extensional part of a deductive database and then performs distributed discovery of approximate global patterns. The discovery process is guided by means of a three stepped strategy. In the first step, the set of original N reference objects is partitioned into n approximately equally-sized subsets ($n << N$). Each partition includes a subset of the reference objects and the set of task relevant objects. In the second step, the frequent pattern computation is parallelized and distributed on n nodes of a Grid platform, one node for each partition. In this way, G-SPADA generates n parallel executions of SPADA at the same time and retrieves local patterns which are frequent in at least one of the data partition. In the third step, G-SPADA approximates the set of globally frequent patterns by merging local patterns discovered in several nodes of the Grid platform.

The basic idea in approximating the global patterns is that each globally frequent pattern must be locally frequent in at least k partition of the original dataset. In the case k is set to 1, this guarantees that the union of all local solutions is a superset of the global solution. However, a merge step with $k = 1$ may generate several false positives, i.e., patterns that result locally frequent but globally infrequent. Hence, value of k should be adequately tuned between 1 and n in order to find the best trade-off between false positive and false negative frequent patterns. The merge step also attempts to approximate values of support for the global patterns starting from the local values of support. Details on the three steps are provided in the next subsections.

3.1 Relational Data Partitioning

G-SPADA pre-processes the deductive database of an event log and completes the description explicitly provided for each example (D_E) with the information that is implicit in the domain knowledge (D_I). An example of this saturation step is provided in Example 5.

Example 5. Let us consider the deductive database:

case(c1). case(c2).
activity(c1,a1). activity(c1,a2). activity(c1,a3). activity(c2,a4).
time(a1,10). time(a2,25). time(a3,29). time(a4,13).
description(a1,create). ...
before(A, B) :- activity(C, A1),activity(C, A2),
time(A1,T1), A1≠ A2, time(A2,T2), T1<T2,
not(activity(C, A), A≠ A1, A≠ A2, time(A,T), T1<T, T<T2)

By performing the saturation step, the predicates *before(a1,a2)* and *before(a2,a3)* are made explicit in the database.

Saturation precedes data partitioning. In this way, redundant inferences are prevented for properties and relations of task-relevant objects shared from two or more reference objects belonging to different data partitions. The definition of reference objects sharing a task-relevant object is provided in the following.

Definition 7. *Given two reference objects $s1$ and $s2$ in D, $s1$ and $s2$ share the task-relevant objects t in D iff there exists a relational pattern p on D such that:*

- *p satisfies the linkedness property,*
- *p involves two variables, say V and Z, such that the former refers to a reference object and the latter refers to some task-relevant object,*
- *there are two substitutions $\theta_1 = \{V \leftarrow r_1, Z \leftarrow t\}$ and $\theta_2 = \{V \leftarrow r_2, Z \leftarrow t\}$, with $r_1 \neq r_2$ such that both $p\theta_1$ and $p\theta_2$ are true in D.*

Example 6. Let us consider the extensional database D:
case(c1). case(c2) case(c3).
activity(c1,a1). activity(c1,a2). activity(c1,a3). activity(c2,a4).
user(a1,u1). user(a2,u1). user(a3,u2). user(a4,u2). ...
The reference objects $c1$ and $c2$ share the task relevant objects $u2$ in D since there exists the pattern P:
P : *case(C), activity(C,A),user(A,U)*
and the substitutions $\theta 1 = \{C/c1, U/u2\}$ $\theta 2 = \{C/c2, U/u2\}$ of p in D.

Data partitioning is performed by randomly splitting the set of reference objects in n approximately equal-sized partitions such that the union of the partitions is the entire set of reference objects. These data partitions are enriched by adding the ground predicates which describe properties and relations of the reference objects falling in the partition at hand. Subsequently, properties and relations of task-relevant objects related to reference objects according to some foreign key path are also added to the partition.

3.2 Distributing Computation on Grid

Each sample dataset is shipped along with the G-SPADA pattern discovery algorithm to computation nodes on Grid using gLite middleware on the INFN (National Institute of Nuclear Physics) Grid. This is done by submitting parametric jobs described in JDL (Job Description Language) through the CLI (command

line interface). Submission of jobs on Grid are divided in several steps. Authenticate on a UI (user interface) through PKI based authentication system with proxy credentials (GSI). Prepare the jobs (JDL, shell script to automate procedure, input file). Upload (Stage-in) a set of dataset. Submit a relative parametric job. Check/wait results. Finally, once the job is executed on Grid, we get the output (Stage-out) files containing the frequent pattern sets along with their support count for each sample.

3.3 Computing Approximate Global Frequent Patterns

The n sets of local frequent patterns are collected from the corresponding computation nodes of the Grid platform and then merged to approximate the set of global patterns. Patterns returned by the merging step are a subset of the patterns potentially discovered by processing the entire dataset.

For each local pattern discovered in at least k data partitions ($1 \leq k \leq n$), G-SPADA derives an approximate of the global support by averaging the support values collected on the n partitions. In the case k is set to n the approximate support corresponds to the true support of the pattern on the entire dataset. The check that the same local pattern occurs in different partitions is based on an equivalence test between two patterns under θ-subsumption, which corresponds to performe a double θ-subsumption test ($P \succeq_\theta Q$ and $Q \succeq_\theta P$). Local patterns occurring in less than k partitions are filtered out. The global frequent patterns obtained following this merge procedure approximate the original frequent patterns which can be possibly mined on the entire dataset. By tuning the parameter k, different results are obtained. An example of global pattern is represented in the following:

case(A), activity(A,B), is_a(B,namemaker), before(B,C), is_a(C,workflow), description(C,workinprogress) [*k=7, avgSup=72.5%*]

which describes the order of execution between two sequential activities, namely B and C, in the process A. B is a name maker activity while C is a workflow activity that is C is described as work in progress. 7 means that this pattern is found in 7 partitions (sample-level support), while 72.5% is the macro average support obtained by averaging the support values computed on the n samples.

4 Experimental Results

Experiments are performed by processing an event log provided by THINK3 Inc(http://www.think3.com/en/default.aspx). THINK3 is one of the most significant global player in Cad and Plm market whose mission is to help manufacturers optimizing their entire product development processes and helps industrial designers creating better, more innovative products.

Multi-level approximate process patterns are discovered from an event log that traces the behavior of 21,256 process executions of one customer of THINK3. The period under analysis is from January 1st to February 28th, 2006. Each

case is traced by registering its events. Each event describes the activity executed within a case and the activity performer. The activities model six tasks (or classes of operations), that is, workflow, namemaker, deleteEnt, prpDelete, prpModify and cast, while the performers denote the category of person (or system) executing the activity, that is, user or administrator.Hence, activities and performers are described by means of three-level hierarchies (see Figure 1). Each hierarchy is mapped into the three granularity levels thus allowing to deal uniformly with both hierarchies at once.

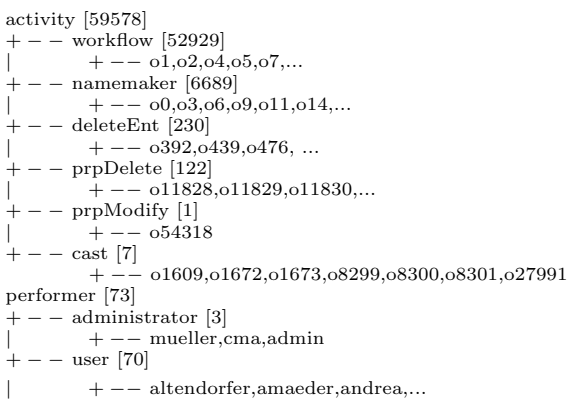

Fig. 1. Three-level hierarchies on activity and performer.

For each activity, a text description of the performed operation is registered in the event log. This text includes a left part and a right one ("left::right"). The right part is a characterization of the description of the operation provided in the left part. Some examples of descriptions registered in the event log are *create::workinprogress*, *t2f::freigabe* and *wip2f::freigabe*. Thirty-six distinct descriptions are registered in the log, but several of them share the same left or right part. For example, *k2f::freigabe, m2f::freigabe, n2f:: freigabe, wip2f::freigabe* share the right part, while *document::doccad, document::doccad3d* share the left part. By interpreting this structure, the activity descriptions is practically boiled down into two predicates, that is, *leftDescription(activity, text).* and *rightDescription(activity, text).* A performer is described by the belonging group. Twenty two distinctive groups are registered in the event log. Performers labeled as users and administrators can eventually belong to the same group.

G-SPADA takes as input a deductive database whose extensional part includes 395,404 ground predicates. Reference objects are the cases, while task-relevant objects are the activities and performers. Temporal autocorrelation of events is taken into account by defining the *before* binary predicate in the intensional part of the deductive database. In this way, an activities of temporally related events are taken into account when investigating properties of each each event and performer. A subset of data is reported in Figure 2.

case(035308). case(035309). ... activity(035308,o535). activity(h10170195,o539). ...
descleft(o535,k2f). descright(o535,freigabe).
descleft(o539,wip2k). descright(o539,konstruktion). ...
time(o525,213183). time(o539,232370). ... user(o535,heide). user(o539,ukoe). ...
group(ukoe,eks_m). group(heide,ekv_j). ...

before(A, B) :- activity(C, A1),activity(C, A2),
time(A1,T1), A1≠ A2, time(A2,T2), T1<T2,
not(activity(C, A), A≠ A1, A≠ A2, time(A,T),T1<T,T<T2)

Fig. 2. Log event representation.

Data are split into twenty partitions. The discovery of the local multi-level frequent patterns is distributed on twenty nodes of the gLite Grid platform. The Grid infrastructure is required to process the huge amount of data registered in the log. Indeed, SPADA generates a memory exception when running on the entire dataset. Multi-level process patterns are discovered at each node with $minsup[l] = 0.2$ ($l = 1, 2$) and max_len_pat=9[5]. For each level of granularity, global patterns are approximated from the local ones by varying k between 1 and 20. The number of discovered global patters are reported in Table 1. Obviously number of global patterns decreases by increasing k.

Table 1. Number of global frequent patterns discovered by varying k in [1,20]

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
#P	428	424	412	412	400	372	372	364	356	356	344	344	314	312	312	311	263	259	259	235

Global patterns provide a compact description of the instances of process traced in the log. They provide a multi-level insight of the order execution and/or organization of the process traced in the log. Some interesting patterns are described below.

At level 1, G-SPADA discovers the process pattern P1, that is:

P1: *case(A),activity(A,B),is_a(B,activity),before(B,C),is_a(C,activity), descright(B,workinprogress),descleft(B,creation), descleft(C,wip2k),descright(C,construction).* [*k=20, avgSup=46.63*]

P1 captures the execution order between two activities (B and C) within a case (A). B is described as a *Work in progress* on a *creation* activity, while C a *wip2k* on a construction operation. By descending one level of the hierarchies defined on activity and performer, G-SPADA discovers the finer grained global relational pattern P2, that is:

P2: *case(A),activity(A,B),is_a(B,workflow),before(B,C),is_a(C,workflow), descright(B,workinprogress),descleft(B,creation), descleft(C,wip2k),descright(C,construction).* [*k=20, avgSup=46.63%*]

[5] max_len_pat is the maximum number of predicates to be included in a pattern.

P2 clarifies that B and D are workflow activities. An approximate of the P2 support is reconstructed from the support of P2 on the partitions where it is locally frequent, that is, P2 is covered by at least 9913 cases registered in the original event log.
The process pattern P4 is reconstructed from local patterns discovered in only three of the split partitions. P3 captures the order of execution between two workflow activities, namely B and C where B is a creation operation performed by an administrator D, while C is a freigabe operation.
P3 is another example of process pattern discovered by G-SPADA.

P3: *case(A), activity(A,B), is_a(B,namemaker), before(B,C),is_a(C,workflow), before(C,D), is_a(D,workflow), descleft(C,creation), descleft(D,wip2k)* [*k=16, avgSup=21.35%*]

P3 describes the execution order that involves three sequential activities, namely B, C and D. B is a namemaker activity, while C and D are workflow activities. C is described as a *creation* operation, while D is described as *wip2k* operation.

5 Conclusions

Process mining targets the automatic discovery of knowledge from massive event logs. This mission adds a substantial complexity to the traditional data mining tasks. Indeed, event logs describe objects of different type which are naturally modeled as several relational data tables. Events have a timestamp that implicitly define a total order and imposes some temporal autocorrelation, that is, the effect of some event at any time may not be limited to the specific event since nearby events (e.g., before event) may present some kind of influence. Activities and performers can be considered at different levels of granularity, that is, they can be organized in hierarchies of classes. Finally, possible domain knowledge (e.g., before relation) cannot be ignored in searching for process patterns.

Although, the multi-relational system SPADA offers a solution to all sources of complexity coming from process mining, SPADA fails in processing massive data such as event log. In order to exploit potentialities of SPADA for process mining, we develop a grid-based solution that allows to approximate the execution of a descriptive task of process mining by distributing multi-level relational frequent pattern discovery on a Grid platform. G-SPADA allows discovering compact descriptions of real process, control, data, organizational, and social structures. At the same time the implementation within a Grid platform allows processing massive event logs. Experiments on the real event log provided by THINK3 allows discovering several human interpretable patterns which captures regularities in the execution of activities and the characteristics of the performers of a business process. Such patterns can be used to deploy new systems supporting the execution of business processes or as a feedback tool that helps in auditing, analyzing and improving already enacted business processes.

As future work, we plan to explore research on ontology in order to enhance expressiveness during constraints definition and querying [7]. Furthermore, we intend to extend empirical evaluation by processing benchmark log data.

Acknowledgments

This work is in partial fulfillment of "TOCAI.it" project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet". The authors wish to thank THINK3 Inc. for having provided process data used in the experiments.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *EDBT*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer, 1998.
2. R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *International Conference on Management of Data*, pages 207–216, 1993.
3. J. E. Cook and A. L. Wolf. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, 1999.
4. S. Džeroski and N. Lavrač. *Relational Data Mining*. Springer-V., 2001.
5. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.
6. E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. *SIGMOD Rec.*, 26(2):277–288, 1997.
7. X. Hou, J. Gu, X. Shen, and W. Yan. Application of data mining in fault diagnosis based on ontology. In *ICITA '05: Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume 2*, pages 260–263, Washington, DC, USA, 2005. IEEE Computer Society.
8. F. A. Lisi and D. Malerba. Inducing multi-level association rules from multiple relations. *Machine Learning*, 55(2):175–210, 2004.
9. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
10. R. S. Michalski. A theory and methodology of inductive learning. pages 323–348, 1993.
11. G. D. Plotkin. A note on inductive generalization. 5:153–163, 1970.
12. C. Silvestri and S. Orlando. Distributed approximate mining of frequent patterns. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 529–536, New York, NY, USA, 2005. ACM.
13. W. M. P. van der Aalst, H. A. Reijers, A. J. M. M. Weijters, B. F. van Dongen, A. K. A. de Medeiros, M. Song, and H. M. W. Verbeek. Business process mining: An industrial application. *Inf. Syst.*, 32(5):713–732, 2007.
14. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.
15. B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The prom framework: A new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.
16. M. J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency*, 7(4):14–25, 1999.