Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 62 (2007) 138–155
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

# Interaction pattern detection in process oriented information systems

Schahram Dustdar [a,b,*], Thomas Hoffmann [a]

[a] Distributed Systems Group, Vienna University of Technology, Institute of Information Systems, Argentinierstrasse 8/1841, A-1040 Wien, Austria
[b] University of Groningen, Department of Mathematics and Computing Science, The Netherlands

Received 20 January 2006; accepted 27 July 2006
Available online 28 August 2006

**Abstract**

Finding interaction patterns is a challenging problem, but this kind of information about processes or social networks might be useful for an organization's management to understand the role of specific persons in processes. Ad-hoc processes are of special interest, because they result from runtime-collaboration between the participants, not using predefined models specifying the persons responsibilities and the order of activities. Because social network analysis (SNA) is closely related to interaction pattern detection, we introduce it as a method to determine properties of social networks like project teams. In order to support the detection of these patterns, we discuss the necessity of additional semantic activity information, and we propose rules and an algorithm that allow detecting such patterns automatically. We apply our algorithm in a case study, using Caramba to perform an example ad-hoc process.


*Keywords:* Interaction patterns; Pattern finding process; Social network analysis; Caramba; Process mining

## 1. Introduction

The competitive pressure in today's economy increases continuously. To stay competitive, there is a need for organizations to optimize their business processes and their intra-organizational communication. Actual work within an organization can deviate from process definitions due to many reasons. One method to improve organizational processes is process mining (e.g., [5,6]). It allows both, identification of processes from transaction logs, and deviation detection between a given process model and real world process executions. The management can use this kind of information to optimize an organization's performance. Additionally to the knowledge about how a process works it is important to understand the communication within an organization, because insufficient communication decreases efficiency. An organization is a social network, and

* Corresponding author. Address: Distributed Systems Group, Vienna University of Technology, Institute of Information Systems, Argentinierstrasse 8/1841, A-1040 Wien, Austria. Tel.: +43 1 58801 18 414; fax: +43 1 58801 18 491.
*E-mail addresses:* dustdar@infosys.tuwien.ac.at (S. Dustdar), thomas.hoffmann@onemail.at (T. Hoffmann).

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.07.010

social network analysis (SNA) [8] can be used for this kind of analysis. It offers the opportunity to determine properties of actors (an organization's employees), groups, or the whole organization. With SNA, for example, it is possible to find out how much a person communicates with others, or if he or she has a central role within the organization. SNA also allows various kinds of analysis, and this information supports the management to initiate improvements. Discovering complex interaction patterns offers additional knowledge about the role of actors within an organization. This information is important, because the more the management knows, the better they can prepare the organization for the future. In particular, interaction pattern detection is important in case of highly dynamic (ad-hoc) processes, because they result from runtime-collaboration between process participants, thus preventing the derivation of the actors' roles solely based on predefined process models. Our main contribution to this area is an algorithm that detects an initial set of interaction patterns within a social network.

This paper is organized as follows. Section 2 lists some related work. Additionally, this section introduces some interaction patterns from the software architecture domain into the domain of business processes. Section 3 contains basic information about SNA. We present some SNA-metrics, systems and applications. We propose an algorithm that enables us to detect interaction patterns by using additional semantic information, some rules, and SNA metrics. The next section deals with different metrics for process mining that are used during construction of social networks. The case study in Section 5 uses a real world business process to show how the pattern finding algorithm works. Furthermore, we analyze properties of the actors with some SNA-metrics. Section 6 concludes the paper and outlines some future work.

## 2. Related work – towards understanding interaction patterns

The main contribution of this paper is the pattern finding algorithm we introduce in Section 3. It is based on knowledge from the process mining and the social network analysis (SNA) domains. Recently, the topic of process mining has been gaining more attention both in practice and research [1,5]. Gartner identifies Business Process Analysis (BPA) as an important aspect of the next generation of BPM products [2]. Note that BPA covers aspects neglected by many traditional workflow products (e.g., diagnosis, simulation, etc.). Business Activity Monitoring (BAM), which can be considered as a synonym to process mining, is named by Gartner as one of the emerging areas in BPA [2]. The goal of BAM tools is to use data logged by the information system to diagnose the operational processes. An example is the ARIS Process Performance Manager (PPM) of IDS Scheer [3]. ARIS PPM extracts information from audit trails (i.e., information logged during the execution of cases) and displays this information in a graphical way (e.g., flow times, bottlenecks, utilization, etc.). Many other vendors offer similar products, e.g., Cognos (NoticeCast), FileNet (Process Analyzer), Hyperion (Business Performance Management Suite), Tibco (BusinessFactor), HP (Business Process Insight), ILOG (Jviews), and webMethods (Optimize/Dashboard). These tools show the practical relevance of process mining. Unfortunately, these tools only focus on measuring performance indicators such as flow time and utilization and do not at all focus on discovering the process and its organizational context. For example, none of the tools mentioned actually discovers causal relations between various events or the underlying social network. Moreover, the focus of these systems is on well-defined processes and they are unable to handle ad-hoc business processes. Note that for ad-hoc business processes it is not sufficient to look into performance indicators such as flow time and utilization, i.e., it is vital to have insight in the actual processes as they unfold, emerge, and/or change. The basis for process mining are workflow logs. Ref. [4] discusses a general workflow log format (XML) that should simplify process mining because mining tools should not have to deal with lots of proprietary log formats. For example, TeamLog [12] is a tool that allows to create XML logs on basis of Caramba's [11] process information. Caramba is one of the few process-aware collaboration systems with good ad-hoc process support. TeamLog accesses Caramba's database and converts its process information to the general workflow log format, thus enabling mining tools to analyze processes performed with Caramba. Unfortunately, this workflow log format does not consider all information required to find the patterns that we will discuss in the remainder of this paper. Therefore, there is a need for proper modifications and enhancements on this workflow log format. In addition to process mining, the idea behind this paper is tightly connected to social network analysis. Wasserman and Faust [8] explain that social network analysis (SNA) focuses on the analysis of relationships among social entities, and on the patterns and implications of these relationships. Our

overall goal is to find interaction patterns in networks. As our first contribution to this domain, we will look for three specific patterns originating from the Software Engineering domain, trying to develop rules (partly based on SNA) and procedures for pattern detection.

In the Software Engineering domain there are some architectural patterns that allow the description of software systems (e.g., [10,13]). We take three of them (proxy, broker and master–slave pattern, Fig. 1) use them as metaphors and introduce them in the domain of business interaction between different parties.

A *proxy* is used as placeholder for another component (the original), i.e., instead of contacting the original directly the client sends its request to this placeholder-component. Additionally to forwarding the client's request to the original and sending back the response, the proxy does some pre- or post-processing depending on its type (remote proxy, protection proxy, cache proxy, synchronization proxy, firewall proxy, etc. [10]). For example, a protection proxy, which is used to protect the original from unauthorized access, checks the client's access authorization before it forwards the request to the original. In most proxy types there is a 1:1 relation between proxy and original (Fig. 1a), i.e., a proxy is a placeholder for exactly one component. But there are two exceptions, remote proxies and firewall proxies, where a proxy is responsible for multiple originals (Fig. 1b, details can be found in [10]). As an example in business practice, we might interpret a secretary as a kind of protection proxy when we focus on incoming requests about meeting schedules. After the secretary has received requests from other business actors who would like to have a meeting with the boss, he/she contacts the boss (if the requestor "is allowed" to get a meeting date), fixes a date, and informs the requesting business actor. Another pattern originating from the domain of software architecture is the *broker* (Fig. 1c). Although it looks similar to the proxy pattern in Fig. 1b, the idea differs. In contrast to a proxy, a broker does not perform any pre- or post-processing. Its major goal is to achieve location transparency of servers/services. A client sends a request to a broker-component which is responsible to locate a server/service that can handle the request. Then the broker forwards the request to the appropriate component, receives its response and delivers the response to the client. To give an example, we can think of a software project, where one person from the customer's project team is responsible for answering questions. When this person receives a request, he has to find and contact the appropriate specialist and sends the answer back to the requesting person (e.g., a software developer working on the requirements analysis). In the literature further subcategories of the broker pattern are discussed (e.g., [10]). Another pattern which can be introduced into the domain of business interaction is the *master–slave* pattern (Fig. 1d). To answer the request, the master distributes the tasks to multiple identical components (slaves) and calculates the response for the client using the slaves' results. Master–slave patterns are used to achieve fault tolerance, parallel processing or to increase accuracy. To enable fault tolerance, the master sends the request to multiple identical slaves and waits for an answer. As long as at least one slave is still working, the client gets an answer for its request. Parallel processing is possible when the master splits the task into multiple identical subtasks, each of them processed by one slave. The master uses all subtasks-results to calculate the response for the client. One example in business practice to increase accuracy is the creation of a cost analysis for a software project. The boss will contact

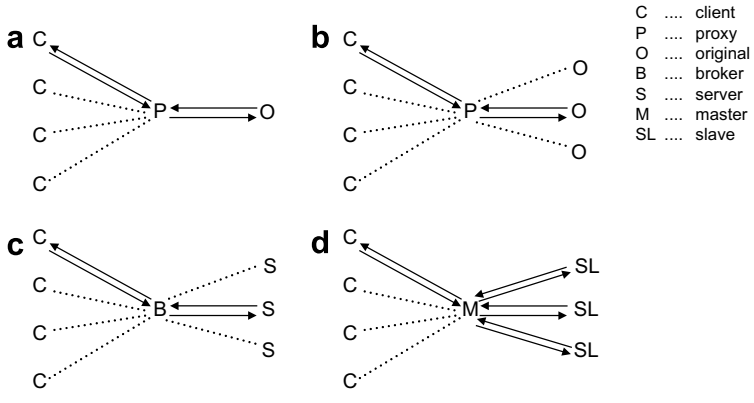


Fig. 1. Proxy (a, b), broker (c) and master–slave patterns (d).

different experienced software developers to collect different opinions. When he receives the results, he compares them and checks if the difference between the estimated costs is high or low. If it is low, i.e., nearly all experienced software developers think about the same costs, it is an indicator for the chief that it is not risky to use this as the basis for an offer.

## 3. Using additional semantic information and SNA metrics to find interaction patterns

Proxy, master–slave, and broker are three interesting patterns in business interactions. Detecting these patterns in real-world business processes would provide an additional level of understanding about the business actors' communication. To allow automatic detection of these patterns, we propose to use combinations of SNA metrics together with some additional rules. It must be clear that these combinations are indicators for the patterns, but it might be possible to construct counterexamples, where the proposed combination is not sufficient to identify the patterns exactly.

First of all, it is important to know that in most cases it would not be possible to detect proxies, brokers, etc. by taking the whole communication into consideration, because in most cases actors are responsible for multiple tasks. We need additional semantic information to detect these patterns automatically. This can be seen in the example communication pattern in Fig. 2. By looking at all communication ties, it is not possible to determine if actor D is a proxy, a broker, a master or none of them. Or it also might be possible that actor D (Fig. 2) serves as proxy for F and as master for E and C at the same time. Therefore, we have to use additional semantic information in the pattern finding process. We need *causal information*, i.e., knowledge about which tasks belong to a specific request. As an example, we do not want to use the communication between F and G for the pattern finding process if it has no causal relation to the request from E to F (Fig. 2). Additionally, we propose the use of *activity categories*. For example, if we know that an actor performs an activity which belongs to the category "security tasks" and we assume that there is a proxy pattern, it would be possible to check if it is a protection proxy or not. Another helpful information is knowledge about the *task–subtask relation*. This information is especially important to identify master–slave patterns for parallel processing, where a master splits a task into different subtasks, sends them to slaves and uses their results to calculate the response for a client. Our last proposal for helpful additional semantic information is to store the *kind of request*. Fig. 2 shows that it is difficult to determine the roles (broker, etc.) of each actor. Information about the kind of request (e.g., "request for meeting date") would help to identify these patterns/roles. For example, if A and B (Fig. 2) send requests of different kinds to actor D, it is unlikely that D is a proxy, but it is possible that it is a broker.

Assuming that this additional semantic information is available during the process of pattern finding, we propose rules that would help to find these patterns (proxy, master–slave, broker). We have to distinguish between a common rule valid for proxy, master–slave and broker-patterns, and specific rules for each of them. We assume that only those communication ties are considered, which have a causal relation to the requests (this filtering must be part of the pattern finding process). When the pattern for a group of actors is checked, other ties (with no causal relation to the requests) are skipped. We use Java syntax/code to illustrate these rules and the pattern finding algorithm.

Each rule is defined in a separate class (Fig. 3), inheriting from a the base class *PatternRule* which implements common behavior and the method signature. The check-method is overloaded in the classes *CommonRule*, *ProxyRule*, *MasterRule* and *BrokerRule*, and it is used to validate the rule against a given social network

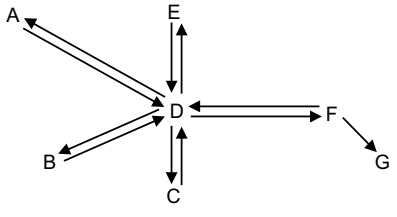


Fig. 2. Communication between multiple actors.

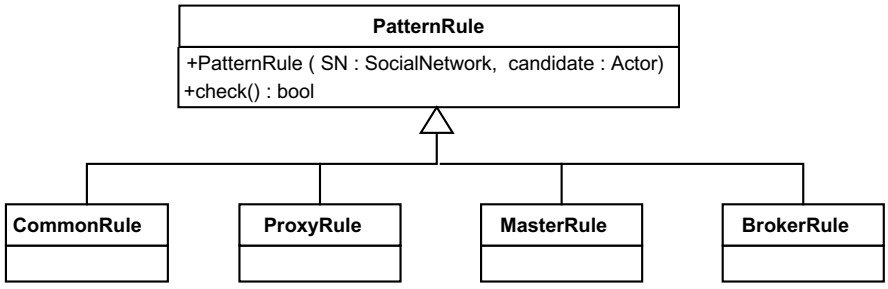


Fig. 3. Classes used to formulate the pattern rules.

and a proxy/master/broker-candidate. Before we explain the pattern finding algorithm, we give more detailed information about each rule.

### 3.1. Listing 1. Common rule

```
class CommonRule - method check()
member candidate: candidate-actor
member SN:        social network of interest

public boolean check()
{
  //check candidate's indegree and outdegree
  if ((SN.indegree(candidate) < 2) ||(SN.outdegree(candidate) < 2))
     return false;

  //Check distance between clients and candidate
  Actor[] clients=SN.get_clients(candidate); //find all clients
  if (clients.length==0) return false;
  for (int i=0;i<clients.length;i++)
     if (SN.distance(candidate,(Actor)clients[i])!=1) return false;

  //Check distance between clients and partners
  Actor[] partners=SN.get_partners(candidate); //find all partners
  if (partners.length==0) return false;
  for (int j=0;j<partners.length;j++)
     if (SN.distance(candidate,(Actor)partners[j])!=1) return false;

  //Check task-independence/degrees
  for (int k=0;k<partners.length;k++)
  {
    Actor curpartner=(Actor)partners[k];
    //if partner performs his task independently
    if (SN.get_actortask(curpartner).is_performed_independently)
    {
      if ((SN.indegree(curpartner)!=SN.weight(candidate,curpartner)
       || (SN.outdegree(curpartner)!=SN.weight(candidate,curpartner))
         return false;
    }
  } //END FOR
  return true;
}
```

This rule (Listing 1) determines that proxies as well as masters or brokers must have a minimum outdegree of 2, because there is at least one partner (original, server, or slave) who gets a request from actor $x$, and there is at least one client receiving a response. The lower limit of actor $x'$ indegree is 2, because he receives at least one request from a client and one response from a partner. The distance between clients and the candidate (actor $x$) has to be 1, as well as the distance between actor $x$ and all partners. If we assume that partners

perform the task of interest independently, i.e., they do not need to contact other parties for task execution, each partner's in- and out-degree is equal to the number of ties from the candidate to the partner, because the partner's communication is limited to receiving and answering the candidate's request. Although it is not reflected in the common rule, we claim that actor $x$ has a high degree centrality and high prestige when there is a large number of clients, because each additional client adds a new inbound and a new outbound tie to actor $x$. Next, we formulate the rule that must hold for proxies.

### 3.2. Listing 2. Proxy-specific rule

```
class ProxyRule - method check()
member candidate: candidate-actor
member SN:        social network of interest

public boolean check()
{
  Actor[] clients=SN.get_clients(candidate); //find all clients
  if (clients.length==0) return false;

  //check if all requests are of the same kind
  String tmp_kind=SN.get_actortask((Actor)clients[0]).kind;
  for (int i=1;i<clients.length;i++)
    if (SN.get_actortask((Actor)clients[i]).kind != tmp_kind)
      return false;

  //check candidate's preprocessing task
  if ((candidate.does_preprocessing()) &&
      (SN.get_actor_preprocessingtask(candidate).category ==
       'splitting task')) return false;

  //check candidate's postprocessing task
  if ((candidate.does_postprocessing()) &&
      (SN.get_actor_postprocessingtask(candidate).category ==
       'calculating_response_from_subresults')) return false;
  return true;
}
```

Listing 2 contains the ProxyRule's check method that implements the proxy-specific rule validation. For a proxy, all client requests must be of the same kind. Additionally, it is allowed that a proxy does some preprocessing before he contacts a partner. But if so, the preprocessing task must not be a splitting task (divides a request into multiple tasks that are forwarded to the partners). Analogous to that, an optional postprocessing task is not allowed to combine different subresults to a final response. The subconditions in the proxy-specific rule concerning pre- and post-processing tasks are used to distinguish between proxies and masters.

### 3.3. Listing 3. Master-specific rule

As well as for proxies, masters (Listing 3) require that all requests are of the same kind. Preprocessing of a client request is only allowed if the preprocessing task is a splitting task. Analogous to that, the category of an optional postprocessing task has to be ''calculating_response_from_subresults''. In case of master–slave patterns for fault tolerance or increased accuracy, the client request is equal to the request sent by the master to all of its slaves. Otherwise, in case of master–slave patterns for parallel processing, there is a task–subtask relation between the client request and the master's requests directed to its slaves. Another subcondition in the master-specific rule states that there are at least two slaves, because a single slave would not be able to implement fault tolerance, increased accuracy, or parallel processing, which are the main applications of a master–slave pattern. Finally, if we assume that slaves perform the task of interest independently, i.e., they do not need to contact other parties for task execution, the slaves are structurally equivalent. This is a logical consequence because in this case all slaves only have ties from and to the master.

```
class MasterRule - method check()
member candidate: candidate-actor
member SN:        social network of interest

public boolean check()
{
  Actor[] clients=SN.get_clients(candidate); //find all clients
  if (clients.length==0) return false;

  //check if all requests are of the same kind
  String tmp_kind=SN.get_actortask((Actor)clients[0]).kind;
  for (int i=1;i<clients.length;i++)
  {
    if (SN.get_actortask((Actor)clients[i]).kind != tmp_kind)
      return false;
  }

  //check candidate's preprocessing task
  if ((candidate.does_preprocessing()) &&
      (SN.get_actor_preprocessingtask(candidate).category !=
       'splitting task')) return false;

  //check candidate's postprocessing task
  if ((candidate.does_postprocessing()) &&
      (SN.get_actor_postprocessingtask(candidate).category !=
       'calculating_response_from_subresults')) return false;

  for (int j=0;j<clients.length;j++)
  {
    Actor curclient=(Actor)clients[j];
    //Activity candidate performs, reacting to request
    //from curclient (client_act)
    Activity cand_act=SN.get_actortask(candidate,curclient);
    Activity client_act=SN.get_actortask(curclient);
    ActivityGroup a_group=SN.get_activity_group();

    //master pattern for fault tolerance and increased accuracy
   //client's request must be equal to request sent from candidate to
   //partner
   if ((!cand_act.is_identical(client_act)) &&
       (a_group.is_task_subtaskrelation(client_act,cand_act)))
     return false;
  }

  //Check number of partners (at least 2)
  Actor[] partners=SN.get_partners(candidate); //find all partners
  if (partners.length < 2) return false;

  //if partner performs his task independently, he must be
  //structurally equivalent to all other partners
  for (int k=0;k<partners.length;k++)
  {
    Actor curpartner=(Actor)partners[k];
    if (SN.get_actortask(curpartner).is_performed_independently())
    {
      for (int u=k+1;u<partners.length;u++)
      {
        Actor tmpact=(Actor)partners[u];
        if (!SN.is_structural_equivalent(curpartner, tmpact))
          return false;
      }
    }
  }

  return true;
}
```

### 3.4. Listing 4. Broker-specific rule

```
class BrokerRule - method check()
member candidate: candidate-actor
member SN:        social network of interest

public boolean check()
{
  Actor[] clients=SN.get_clients(candidate); //find all clients
  if (clients.length==0) return false;

  //check if requests of the same kind are forwarded to the same
  //partner
  for (int i=0;i<clients.length;i++)
  {
    Actor curclient=(Actor)clients[i];
    for (int j=i+1;j<clients.length;j++)
    {
      if (SN.get_actortask(curclient).kind==
          SN.get_actortask((Actor)clients[j]).kind)
      {
        Actor p2=SN.get_contacted_partner((Actor)clients[j]);
        if (SN.get_contacted_partner(curclient).compareTo(p2)!=0)
          return false;
      }
    }
  }

  return true;
}
```

If a candidate acts as a broker (Listing 4), we assume that client requests of the same kind are forwarded to the same server. That implies that a broker can receive requests of different types. In contrast to proxies or masters, the broker pattern allows neither pre- nor post-processing.

As well as for the rules, we use Java code to describe the pattern finding algorithm. Fig. 4 shows the classes used for the implementation: *SocialNetwork*, *ActivityGroup*, *Actor*, *Activity* and *Relation*. Table 1 contains more detailed information about these classes and their methods.

We propose a sequence of actions for the *pattern finding process*, that enables the detection of proxy, master–slave, and broker patterns (Listing 5). The input for this algorithm is a workflow log. Based on this

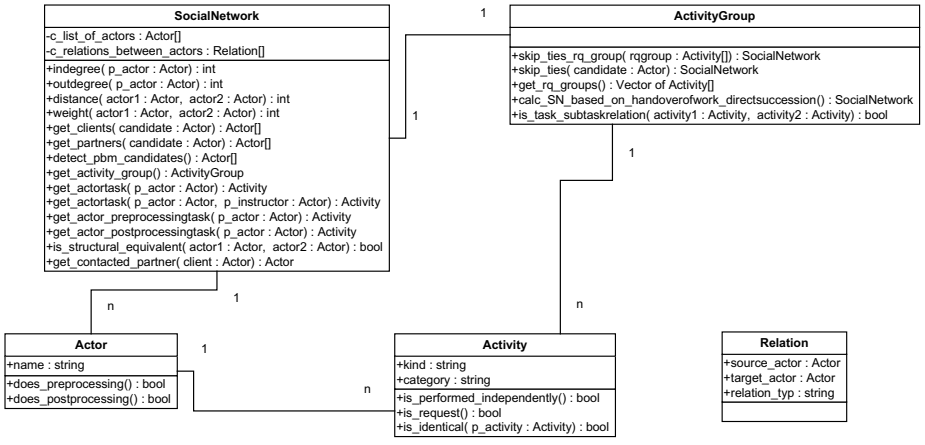


Fig. 4. Classes used to implement the pattern finding algorithm, apart from the pattern rule classes.

Table 1
Class/method description

| Class/method | Description |
|---|---|
| Actor | Process participant |
| does_preprocessing | Checks if the actor does some kind of preprocessing before he performs his activities |
| does_postprocessing | Checks if the actor does some kind of postprocessing before he performs his activities |
| Activity | Activity (one step in the process) |
| Kind | Stores the kind of request (e.g., "request for meeting date") |
| Category | Stores the activity's category (e.g., "security task") |
| is_performed_independently | Checks if the activity is performed independently, i.e., the performer does not need results from others to finish activity execution |
| is_request | Checks if the activity is a request. A request is an activity that causes other process participants to do something, e.g., "instruct to create cost analysis" |
| is_identical | Checks if two activities are "identical". In the algorithm this is used to compare the client's request with the activities "sent" from the candidate to the partners |
| Relation | Represents the relation between actors in the social network |
| source_actor, target_actor | Actors |
| relation_typ | Type of relation, e.g., "handover of work – only direct succession" |
| ActivityGroup | Group of activities (process or part of a process). This datastructure is generated by interpreting the contents of workflow logs. |
| get_rq_groups | Returns groups of requests in the network. All requests within a group are of the same kind. |
| skip_ties | Removes all ties from the network of activities that are not causally related to any of the clients' requests. Returns the resulting subnet |
| skip_ties_rq_group | Removes all ties from the network that are not causally related to the requests within the given request group. Returns the resulting subnet |
| calc_SN_based_on_ handoverofwork_ directsuccession | Creates a social network based on the "handover-of-work" metric (considering only direct succession). |
| is_task_subtaskrelation | Checks if there is a task–subtask relation between two activities |
| SocialNetwork | Social network focusing on the relation between actors |
| indegree | Indegree of a given actor in the network |
| outdegree | Outdegree of a given actor in the network |
| distance | Distance between two actors in the network |
| weight | Number of communication ties between two actors |
| get_clients | Actors serving as clients that send requests to a given candidate |
| get_partners | Actors that are contacted by the candidate (to generate an answer for the client requests) |
| detect_pbm_candidates | Returns all actors that have inbound and outbound communication to the same actors (clients). These persons are potential candidates for a proxy, master or broker |
| get_activity_group | Activity group that was used to create this social network |
| get_actortask | *signature 1 (one actor as parameter)*: returns the task of the given actor. If the actor performs more than one task in the social network, the "earliest" task is return. This method should be used only if the situation assures that the actor performs only one task in this social network<br>*signature 2 (two actors as parameter)*: returns the task of an actor assigned by another actor |
| get_actor_preprocessingtask | Returns the preprocessing-activity an actor performs before he starts execution of his activities |
| get_actor_postprocessingtask | Returns the postprocessing-activity an actor performs after he has executed his activities |
| is_structural_equivalent | Determines if two actors are structurally equivalent in this social network |
| get_contacted_partner | Returns the partner that is contacted by the candidate, caused by a request of the given client actor |

log a process flow must be generated, where nodes are actors and the ties between them represent handover-of-works, i.e., its structure is similar to a social network based on handover-of-work, but there is still information for each single communication tie available. First, the process must detect all nodes that have inbound and outbound communication to the same node (step number 1 in Listing 5). These nodes are candidates for proxies, masters, or brokers. After this identification task each candidate has to be processed separately (step 2). Before we can do further checks, it is necessary to find all ties that are caused by the

client requests to this candidate actor, and all other ties must be skipped (step 3). When this filtering-process is finished, the pattern finding process has to test the common rule (step 4), which must hold for each of the patterns. If this check fails, the candidate actor is neither a proxy nor a master or a broker, and therefore, the process can look for the next candidate actor. Next, if the common rule is fulfilled, the algorithm tries to find out if the candidate actor is a proxy or a master. Proxy and master patterns require that only requests of the same kind are considered. Because in networks a candidate can, for example, act as proxy and master, the algorithm looks for requests of the same kind and builds appropriate request groups (step 5). Each request group is considered separately (step 6). Ties caused by requests outside of the request group are eliminated (step 7). Hence, the proposed algorithm is able to identify proxies (step 8) and masters (step 10) also if they do not act purely as proxy or master. For example, if a person acts as proxy for one person and, additionally to that, as master in another context, the algorithm identifies both roles. After checking for proxy and master, the next step is to check whether the candidate acts as broker (step 13). In contrast to proxies and masters, where groups of similar requests are handled separately (step 6), the algorithm is only able to identify a broker role if the candidate acts purely as broker. Otherwise (e.g., if the candidate does some preprocessing of certain requests which is not allowed in a broker pattern), the pattern is not recognized. The algorithm maintains (steps 9, 11, 14) and prints the list *pbm_list*, which holds the actual roles of each candidate.

### 3.5. Listing 5. Pattern finding algorithm

```
public class FindPatterns
{
  public static void main(String[] args)
  {
    //create ActivityGroup based on workflow-logs
    ActivityGroup a_group= ......

    SocialNetwork net=
      a_group.calc_SN_based_on_handoverofwork_directsuccession();

    Vector result_list=find_patterns(net);

    //Print result
    for (int i=0;i<result_list.size();i++)
    System.out.println((String)result_list.get(i));
  }

  public static Vector find_patterns(SocialNetwork SN)
  {
     //Input: SN...network
     //Output: pbm_list...list of actors who are proxies, masters, or
                          brokers
    Vector pbm_list=new Vector();
1   Actor[] candidatelist=SN.detect_pbm_candidates();

    //Go through candidates
2   for (int i=0;i<candidatelist.length;i++)
    {
      Actor candidate=(Actor)candidatelist[i];
3     SocialNetwork cand_subnet=
                  SN.get_activity_group.skip_ties(candidate);
      //"Create" common rule
      CommonRule common_rule=new CommonRule(cand_subnet,candidate)
4     if (!common_rule.check())
        continue; //check next candidate actor

      Vector requestgroups_same_kind=
5           cand_subnet.get_activity_group.get_rq_groups();

      boolean cand_proxy=false;
      boolean cand_master=false;

      //Go through each requestgroup (contains requests of
      //the same kind
```

```
6       for (int j=0;j<requestgroups_same_kind.size();j++)
        {
          SocialNetwork rq_subnet=
7           cand_subnet.skip_ties_rq_group(requestgroups_same_kind[j]);

          //"Create" proxy rule
          ProxyRule proxy_rule=new ProxyRule(rq_subnet,candidate)
8         if ((!cand_proxy) && (proxy_rule.check()))
          {
9           pbm_list.add('proxy: ' + candidate.name);
            cand_proxy=true;
          }

          //"Create" master rule
          MasterRule master_rule=new MasterRule(rq_subnet,candidate)
10        if ((!cand_master) && (master_rule.check()))
          {
11          pbm_list.add('master: ' + candidate.name);
            cand_master=true;
          }

12        if ((cand_master) && (cand_proxy)) break;
        }  //END FOR requestgroups

        //"Create" broker rule
        BrokerRule broker_rule=new BrokerRule(cand_subnet,candidate)
13      if (broker_rule.check())
14          pbm_list.add('broker: ' + candidate.name);
      }  //END FOR candidates
    }
}
```

## 4. Metrics for process mining

Social network analysis is based on social network data, mostly represented by sociomatrices (matrix) or sociograms (graph). During process mining, this information is collected from workflow logs. Similar to the analysis of social networks (a lot of SNA-metrics exist), different metrics can be used to construct such network data from the underlying workflow log information. The metric that is chosen determines the semantic of the constructed network data. Van der Aalst and Song developed some metrics [9] which can be used to establish relationships between individuals from workflow logs. The mining results are meaningful sociograms which can be further analyzed by means of social network analysis. Fig. 5 gives an overview about the mining metrics developed by Van der Aalst and Song.

In [9] they distinguish between metrics based on (possible) causality, metrics based on joint cases, metrics based on joint activities, and metrics based on special event types. Each of these metrics results in data structures (e.g., sociogram) that can be analyzed using existing SNA tools. The *metric based on (possible) causality* is based on the idea that performers are related if there is a causal relation through the passing of work from one performer to another. Subcategories are handover-of-work and subcontracting. Van der Aalst and Song define that there is a handover of work from individual *i* to individual *j* if there are two subsequent activities where the first is completed by *i* and the second by *j*. The idea of subcontracting is that the workflow log is analyzed with regard to subcontracted work, i.e., mining considers only activities of a person *j* that are performed between two activities of another person *i*. For both handover-of-work and subcontracting refinements are possible. First, the degree of causality can be taken into account (the length of the handover), so that not only direct succession is considered. Another refinement is that multiple transfers within a case can be ignored or not. The third refinement mentioned by Van der Aalst and Song deals with the possibility that only real causal dependencies are taken into account (this refinement requires a process model). Because of these three refinements there are eight variants for both handover-of-work-metrics and subcontracting-metrics. If *metrics based on joint cases* are used, causal dependencies are ignored. Instead, it is counted how frequently two individuals are performing activities for the same case. This may be an indicator of a stronger relation than only "working together". The last category of metrics proposed by Van der Aalst and Song are *metrics based on special event types*. Workflow logs typically contain event-information for an activity

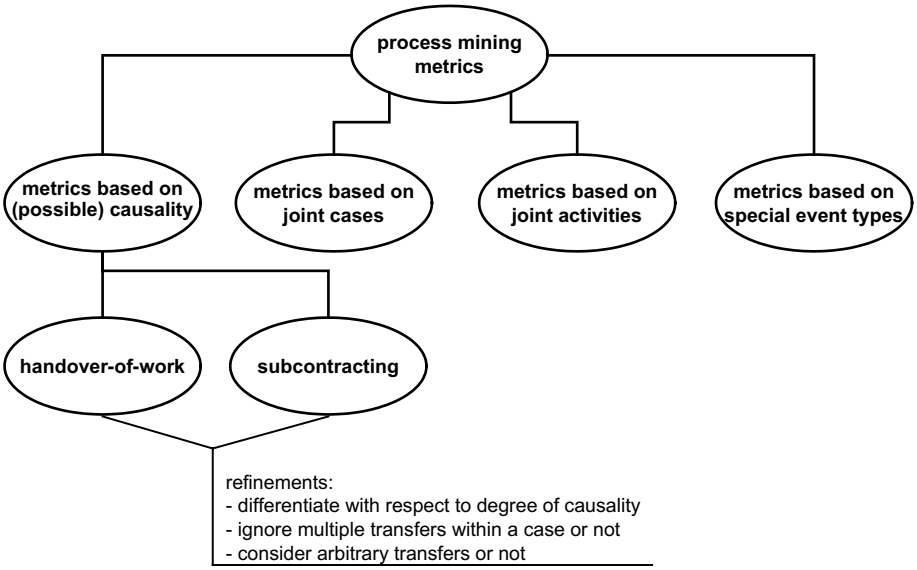


Fig. 5. Mining metrics developed by Van der Aalst and Song [9].

(when the activity was started, when it was delegated to another person, when it was completed, etc.). The idea of this type of metrics is to consider only special event types like ''delegation''. For example, if a person frequently delegates work to another person but not vice versa, there is probably a hierarchical relation between these persons. The idea of *metrics based on joint activities* is to count the number of times a person executes specific activities. In this case, the result of process mining is a performer-by-activity matrix, which contains ''profiles'' of individuals (how frequently they perform specific activities). Then these profiles can be analyzed further by measuring the ''distance'' between the profiles (to quantify how ''equal'' the work of different performers is). Van der Aalst and Song propose distance-metrics like the Minkowski-distance or Hamming-distance to measure the distance between profiles.

## 5. Case study: mining interaction patterns in Caramba

Our case study's basis is a real-world example process for installing new branches (two banks). Firstly, we will look for interaction patterns in the intra-organizational communication. Then some SNA-metrics from Section 3 are evaluated using Agna.[1]

The idea for our case-study example (Fig. 6) comes from [12]. We extend the example by adding communication between our example organization (further called ''J&I'') to external parties (customer, supplier). Additionally, the complexity of J&I's internal communication is increased to enable meaningful SNA and to present different kinds of patterns. The core idea of the example is that two banks form an alliance, whose purpose is to build new branches at lower costs. This can be achieved because hardware/software or real estate are cheaper if a larger amount is ordered. Therefore, the customers instruct J&I together to save money. J&I is responsible to fulfill all common and customer-specific requirements.

### 5.1. Process description

This example involves different parties (Table 2): a customer alliance, two suppliers (estate agent and hardware/software-supplier) and nine employees of J&I, who have to realize the project.

[1] See http://www.geocities.com/imbenta/agna/.

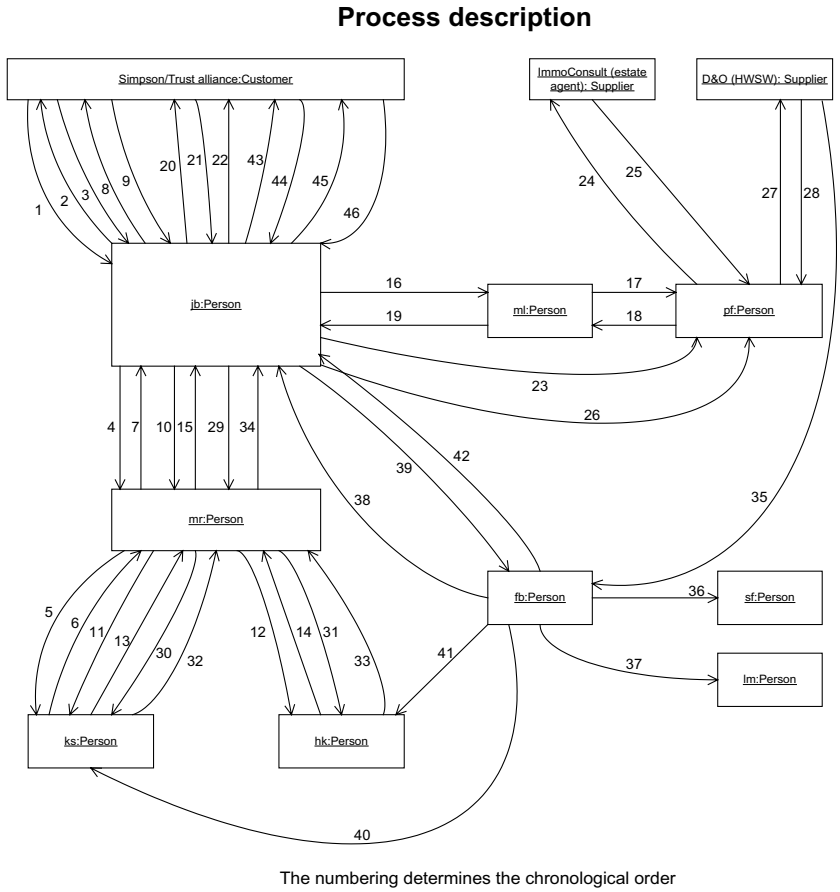


Fig. 6. Example process.

Table 2
Involved parties

| Abbreviation | Full name | More information |
|---|---|---|
| *jb* | Joe Baker | Org. department J&I |
| *mr* | Martin Roth | IT (technical engineer) J&I |
| *ks* | Kurt Schmitt | IT (technical engineer) J&I |
| *hk* | Hans Koller | IT (technical engineer) J&I |
| *fb* | Frank Baumann | IT (technical engineer) J&I |
| *ml* | Monika Lachs | Sales department J&I |
| *pf* | Peter Fogosch | Sales department J&I |
| *sf* | Susan Francis | IT (trainee) J&I |
| *lm* | Lance Manto | IT (trainee) J&I |
| | Simpson/Trust alliance | Customer |
| | ImmoConsult | Estate agent (supplier) |
| | D&O | Hardware/software supplier |

The customer alliance sends a request for an offer (1) to *jb* (project leader). This request contains a short description of the requirements for branches that must be ''built''. *jb*'s requests more detailed information about the requirements (2) because he is not able to send an offer without that. For example, *jb* needs information about the preferred location, size, amount, and location of workstations in the branches, operating systems, applications, expectations about the project's time plan, etc. The customer alliance answers with an

enhanced requirements document (3), that contains customer-independent (valid for Simpson and Trust) as well as customer dependent requirements. *jb* forwards the requirements document to *mr* (4) who should check it for consistency/discrepancies/open issues. *mr* collects questions and also requests comments from *ks* (5, 6). *mr* adds *ks*' answers to the list of questions about requirements and sends it back to *jb* (7), who forwards the questions to the customer alliance (8). After *jb* receives the modified requirements document (9), he instructs *mr* to create a cost analysis (10) which will form the basis for a binding offer. *mr* delegates (11, 12) this task to *ks* (analysis of common and Trust-specific requirements) and *hk* (Simpson-specific requirements). They send the requested analysis back (13, 14) to *mr* who forwards it to the project leader (*jb*) (15) after he has created a common cost analysis document. Because *jb* needs an offer which considers the costs and an extra charge (for profit), he sends the cost analysis to *ml* (16) and expects a complete offer. *ml* does not create the offer by herself. Instead of that, she removes some technical information from the cost analysis and delegates the task to *pf* (17). After he finished this task, *pf* sends the offer back to *ml* (18), who adapts the document formatting and delivers the offer to *jb* (19). *jb* signs the offer and forwards it to the customer alliance. This alliance has to check (20) if it accepts the offer or if it will contact another organization. In our case, both banks of the alliance accept the offer and send an order (21) to the project leader *jb*. This order serves as contract between the customer and J&I. Just for information, *jb* responds with an order confirmation (22).

One of the next steps required to "build" new branches is to find appropriate real estates. Therefore, *jb* instructs *pf* (23) to contact ImmoConsult (24), J&I's default estate agent. This results (25) in two different contracts for the two banks, caused by different requirements: a contract of sale for Simpson Bank and a hire contract for Trust Bank. The next step is to order the hardware/software that is needed to do the IT-installations in both new branches. This is also in *pf's* responsibility (26–28). Because a technical installation plan for the branch installations is still missing, *jb* instructs mr to create such a plan (29). *mr* delegates this task to *ks* (plan for Trust bank) and *hk* (plan for Simpson bank) who send the requested plan back to *mr* (and therefore, indirectly to *jb*) (30–34). After a few days, D&O delivers the ordered hardware/software to *fb* (35). To prepare the workstations for the branches, *fb* expects the trainees (*sf*, *lm*) to do some pre-installations-tasks (installing the operations system, default applications, etc.) which will save time on the scenes of action, the branches (36, 37). After the pre-installation is finished, *fb* informs the project leader *jb* (38). And therefore, *jb* instructs *fb* to initiate the IT-installation in both new branches (39). *fb* fulfils the instruction together with *ks* and *kh* (40, 41) and, after completion, informs the project leader (42). The project leader tells the customer alliance that installation is complete and that they can do their inspection tests (43). After successful tests the customer alliance accepts the work by sending an acceptance confirmation back to the *jb* (44). Finally, the project leader *jb* creates an invoice and the customer initiates the corresponding payment (45, 46).

### 5.2. Locating interaction patterns

The first step in our case study is to locate interaction patterns in the intra-organizational interaction, i.e., we exclude the "external actors" (customer, suppliers) and consider only the interaction between employees of J&I. Hence, we do not consider the corresponding communication ties (1–3, 8, 9, 20–22, 24, 25, 27, 28, 43–46) (Fig. 6). We use the proposed rules and the pattern finding algorithm from Section 3 to find the patterns that we have introduced into the business interaction domain. First of all, the algorithm looks for persons who could possibly act as proxy, master, or broker (identifying candidates, Listing 5, step 1). The resulting set of candidate actors may be quite large, depending on the structure of the social network of interest. In our example, the candidate list will contain at least the actors *ml* and *mr*. Let us assume that the algorithm considers candidate *ml* first.

Fig. 7a contains the social network area of interest around candidate actor *ml*. To be able to test the rules introduced in Section 3, the pattern finding algorithm removes all ties that are not causally related to the requests (Listing 5, step 3). The next step is to test the common rule, which must hold if *ml* acts as proxy, master, or broker (step 4). This is the case, because of the following facts: *ml*'s outdegree = 2, *ml*'s indegree = 2, actor distance between client (*jb*) and *ml* = 1, actor distance between *ml* and partner (*pf*) = 1. Because *pf* performs its task 17 independently (i.e., without contacting other parties), he must have an indegree of 1. Hence,

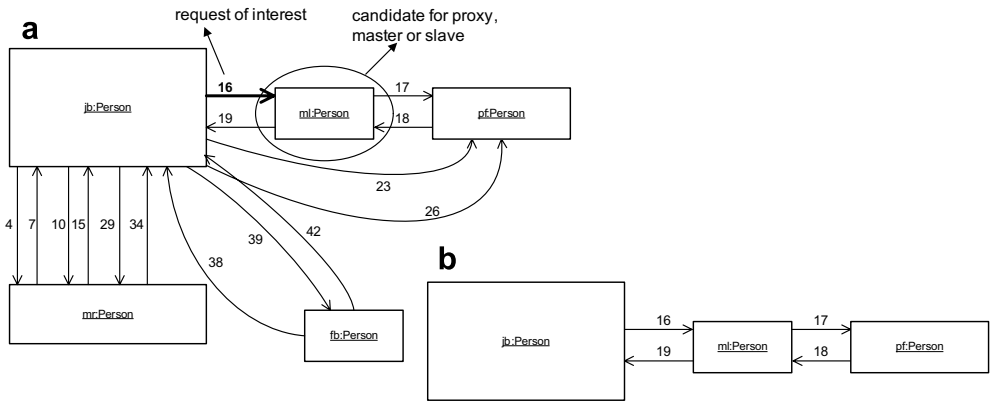


Fig. 7. (a) Network area of interest around candidate actor *ml*, (b) after all ties have been removed that are not causally related to request 16.

the common rule holds. The next step is to verify if *ml* is a proxy or a master. To reach this, the algorithm summarizes requests of the same kind into groups (step 5) and checks each request group separately (step 6). In case of *ml*, this is trivial, because there is only on request (Fig. 7a, 16), and therefore, step 7 does not change the subnet. With these steps, the algorithm has created a subnet (Fig. 7b), which will be used to verify the proxy-specific rules (step 8) and the master-specific rules (step 10). *ml* acts as proxy, because all requests are of the same kind (trivial, because there is only one request), the preprocessing tasks (''removing some technical information from cost analysis'') category is not ''splitting task'', and the postprocessing task (''adapting document formatting'') does not merge different responses from partners to generate the response for *jb*. The master check is negative, because there is only one partner, the pre- and post-processing tasks category does not match the master requirements, the number of partners is smaller than 2, etc. The last step in the algorithm is to verify if *ml* acts as broker (step 13). But pre- and post-processing is not allowed for a broker. Hence, the algorithm classifies *ml* as proxy and continues with the next candidate actor (*mr*, step 2).

Again, all ties that are not causally related to the requests (4, 10, 29) are removed from the social network (Fig. 8b). Then, the common rule test (step 4) is performed: *mr*'s indegree = 8, outdegree = 8, distance between the client (*jb*) and *mr* = 1, distance between *mr* and the partners (*ks*, *hk*) = 1, in-/outdegrees of *ks* (3) and *hk* (2) agree with the requirements of the common rule (both *ks* and *hk* perform their tasks independently). After that, the requests are grouped so that the requests within a group are of the same kind. In our

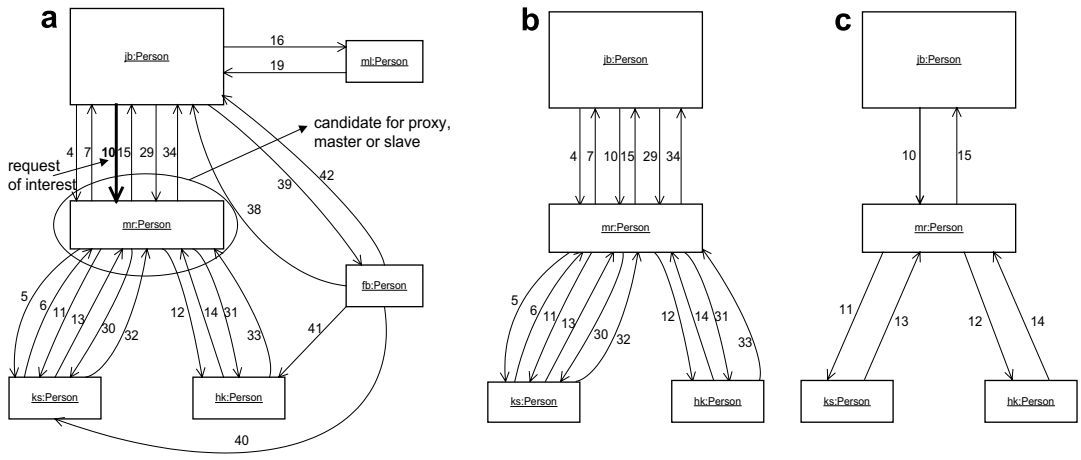


Fig. 8. (a) Network area of interest around candidate actor *mr*, (b) after all ties have been removed that are not causally related to any request, (c) considers only one kind of requests.

case, all request kinds (4, 10, 29) are different and therefore, each request is handled separately in the pattern finding process (steps 5 and 6). In this case study, we will neglect the requests 4 and 29 and focus on request 10. The algorithm removes all ties from the subnetwork (Fig. 8b) that are not causally related to request 10 (step 7). The resulting subnet, which is used to verify the proxy- and master-specific rules, is shown in Fig. 8c. *mr*'s preprocessing task is of category ''splitting task'', because it splits the request from *jb* into 2 subrequests (''analysis of common and Trust-specific requirements'', ''analysis of Simpson-specific requirements''). *mr* merges their answers (postprocessing) to generate the response for *jb*. Because of this pre- and post-processing, the proxy-check (step 8) fails. There is a task–subtask-relation between the requests from *jb* to *mr* and the subrequests sent from *mr* to his partners. Furthermore, there is more than one partner (*ks* and *hk*). Hence, the algorithm classifies *mr* as master. Finally, the pattern finding algorithm tests if *mr* acts also as broker. But that's not the case, because in the rules we propose a broker is not allowed to do any pre- or post-processing.

### 5.3. Applying SNA metrics

The second step in our case study is to apply some SNA metrics to our example workcase. For this reason, we perform the workcase in Caramba and use TeamLog [12] to generate an XML workflow log. Then the mining tool MiSoN [9] is used to create a social network, based on the handover-of-work metric, which is used as input for the SNA-tool Agna. Fig. 9 shows the social network that results from performing the example workcase.

However, the workflow log generated by TeamLog must be modified in some way before it can be used for further processing (MiSoN). There are two reasons for that. Firstly, TeamLog generates additional start- and end-loglines for each workcase, which falsify the result of MiSoN. Therefore, these loglines must be removed manually from the XML workflow log. The second problem is the asynchronous structure of our example workcase. Because we focus only on intra-organizational interactions, 34 and 36 are neighbors in the XML workflow log. However, there is no direct causal relation between these activities, because 36 is initiated by an asynchronous external event (reply from hardware/software supplier). Without proper manual modifications, the resulting social network based on handover-of-work would assume a tie between *mr* and *fb*, because mining tools still have problems to mine such asynchronous situations.

Each communication tie has a value indicating the intensity of communication between the corresponding actors. For simplicity, we removed these values from Fig. 9. Agna can be used to analyze different aspects of this social network: number of nodes and edges, diameter, geodesics between pairs of actors, nodal degrees (indegree, outdegree), centrality (e.g., Bavelas-Leavitt, closeness, betweenness), etc. In our example, Agna reports a diameter of 4 and a density of 0.22 (=number of ties/number of possible ties = 16/72), i.e., that the largest distance between two actors is 4 and that about a fifth of possible communication ties actually appear in this network. Additionally, we used Agna to check centrality metrics to determine the ''most central'' actors in this network. The result was that Joe Baker is the actor with the highest centrality (Bavelas-Leavitt index = 6.92) followed by Martin Roth and Frank Baumann (both 5.63). The actor with the lowest

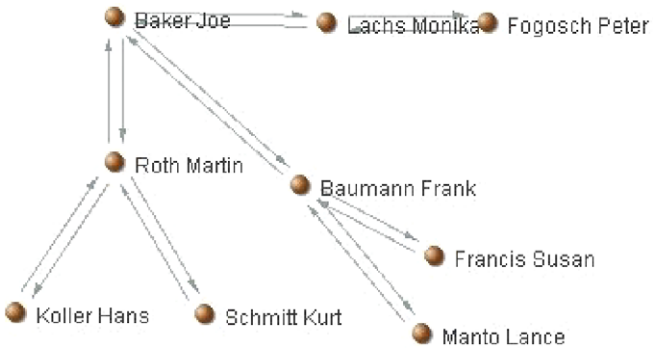


Fig. 9. Social network visualized with Agna.

centrality is Peter Fogosch (3.6). Joe Baker is the most central actor because he has the most outgoing communication ties.

Finally, we conclude our case study by summarizing its main ideas. Firstly, we explained a relatively complex real world example process. In a next step, we tried to locate the interaction patterns that we have introduced from the software architecture domain in the intra-organizational interaction of our example process by using the algorithm we proposed in Section 3. At the end, we applied some SNA metrics to our example workcase.

## 6. Conclusion

The underlying assumption of this paper is that important knowledge can be extracted from social networks. Our intention is to provide an organization's management with information that helps to improve the organization's competitiveness. We introduced interaction patterns (proxy, master–slave, broker) from the software architecture domain in the domain of business processes, because their appearance in an organization's social network provides information about the role of individuals. Our core contribution includes an algorithm capable of detecting these patterns (proposed in Section 3). We presented the pattern finding algorithm based on rules and we emphasize the need of additional semantic information in a business process (causal information about requests, activity categories, task–subtask relations, information about the kind of request). To show how our algorithm works, we applied it to a real world example in our case study. We believe that an organization's management can benefit from social network analysis as well as from identifying interaction patterns to optimize the organizations effectiveness. Teams can be structured more efficiently and effectively if the management knows how the persons work and collaborate. For example, if two persons shall work together but both of them act as proxy, this may cause significant problems in the process. Our algorithm contributes to that goal, allowing to detect these problems. Additionally, the information gathered by the pattern finding algorithm can be used as basis for process optimizations.

Our future work includes applying our algorithm for analyzing the relevance of proxy, master–slave, or broker patterns in ''networks of Web services'', i.e., if the patterns occur during Web service communication [7].

## References

[1] W.M.P. van der Aalst, A.J.M.M. Weijters (Eds.), Process Mining, Special Issue of Computers in Industry, 53(3), Elsevier Science Publishers, Amsterdam, 2004.
[2] Gartner. Gartner's Application Development and Maintenance Research Note M-16-8153, The BPA Market Catches another Major Updraft, 2002. Available from: <http://www.gartner.com>.
[3] IDS Scheer. ARIS Process Performance Manager (ARIS PPM), 2002. Available from: <http://www.ids-scheer.com>.
[4] B.F. van Dongen, W.M.P. van der Aalst, EMiT: A Process Mining Tool, in: J. Cortadella, W. Reisig (Eds.), Application and Theory of Petri Nets 2004, Lecture Notes in Computer Science, vol. 3099, Springer-Verlag, Berlin, 2004, pp. 454–463.
[5] W.M.P. van der Aalst, B.F. von Dongen, J. Herbst, L. Maruster, G. Schimm, A.J.M.M. Weijters, Workflow mining: a survey of issues and approaches, Data and Knowledge Engineering 47 (2) (2003) 237–267.
[6] W.M.P. van der Aalst, A.J.M.M. Weijters, Process mining: a research agenda, Computers in Industry Journal 53 (3) (2004) 231–244.
[7] S. Dustdar, R. Gombotz, Discovering web service workflows using web services interaction mining, International Journal of Business Process Integration and Management (IJBPIM), forthcoming.
[8] S. Wasserman, K. Faust, Social Network Analysis: Methods and Applications, Cambridge University Press, Cambridge, 1994.
[9] W.M.P. van der Aalst, Minseok Song, Mining social networks: uncovering interaction patterns in business processes, in: International Conference on Business Process Management, 2004.
[10] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, Pattern-Oriented Software Architecture – A system of patterns, John Wiley & Sons, 1996.
[11] S. Dustdar, Caramba – a process-aware collaboration system supporting ad hoc, Distributed and Parallel Databases 15 (1) (2004) 45–66.
[12] S. Dustdar, T. Hoffmann, W.M.P. van der Aalst, Mining of ad-hoc business processes with TeamLog, Data and Knowledge Engineering 55 (2) (2005) 129–158.
[13] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns: elements of reusable object-oriented software, Addisson-Wesley, 1995.

**Schahram Dustdar** is Full Professor at the Distributed Systems Group, Information Systems Institute, Vienna University of Technology (TU Wien) where he is director of the Vita Lab. He is also an Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands. He co-authored more than 120 publications in journals, conferences and book chapters. More information can be found at: www.infosys.tuwien.ac.at/Staff/sd.

**Thomas Hoffmann** graduated at the Distributed Systems Group, Vienna University of Technology. His research interests include process mining and Web services.