How to Make Business Processes "Socialize"?

Zakaria Maamar
Noura Faci
Ejub Kajan
Sherif Sakr
Mohamed Boukhebouze

See next page for additional authors

Follow this and additional works at: https://zuscholars.zu.ac.ae/works

Part of the Business Commons, and the Computer Sciences Commons

Recommended Citation
Maamar, Zakaria; Faci, Noura; Kajan, Ejub; Sakr, Sherif; Boukhebouze, Mohamed; and Barnawi, Ahmed, "How to Make Business Processes "Socialize"?" (2015). All Works. 1890.
https://zuscholars.zu.ac.ae/works/1890

This Article is brought to you for free and open access by ZU Scholars. It has been accepted for inclusion in All Works by an authorized administrator of ZU Scholars. For more information, please contact Yrjo.Lappalainen@zu.ac.ae, nikesh.narayanan@zu.ac.ae.
This paper presents an approach that builds upon social computing principles to make business processes “socialize”. First the approach identifies the main components of a business process that are task, person, and machine. A task is a work unit that forms with other tasks a business process and that a person and/or machine execute. Afterwards the approach enriches a business process with details captured from the (execution and social) relations that connect tasks together, persons together, and machines together. While execution relations are widely reported in the literature, there is a growing interest in studying the role of social relations in business processes. The approach uses social relations to build configuration network of tasks, social network of persons, and support network of machines. These networks capture the ongoing interactions that arise when business processes are executed. A system illustrating how these networks are developed is also demonstrated in the paper.

1. Introduction

Social software, exemplified by Web 2.0 applications like social networks, Wikis, and blogs, has forced companies to review their ways of doing business. Many, if not all, companies have an online social presence so they can reach out to more customers, open up new communication channels with stakeholders, and also, showcase their embracement of the latest IT advances and gadgets [1]. Many companies recognize the need of rethinking their strategies and reevaluating their operation models as the world is getting more “social” [2].

Despite the Web 2.0 “fever”, a recent study by Gartner reveals that “...many large companies are embracing internal social networks, but for the most part, they’re not getting much from them” [3]. Social software does not work like an enterprise resource planning application where procedures are defined and employees are told to comply with these procedures. Employees’ commitments (also participation in [1]) are critical to the success of social software, i.e., employees must opt-in rather than be forced [5]. On top of employees’ commitments we argue that other elements contribute to this success, for instance (i) establishing guidelines and techniques to assist IT practitioners integrate social elements into business processes and (ii) demonstrating the social software’s benefits through tangible results (e.g., number of new customers attracted because of a Facebook campaign).

©2015 Zakaria Maamar et al., licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.
According to Gartner narrowing down the social-software view to social networks, only does not shed the light on other systems like business process management systems that exhibit some social aspects [2]. Various interactions take place during the execution of business processes, so we map some of these interactions onto specific social relations between these processes’ components. We refer to these components as task, machine, and person. Indeed tasks are put together to form processes, persons collaborate together on complex tasks, and machines replace each other in the case of failure, are examples of social relations that business process management systems exhibit and hence, can be captured. While we acknowledge that tasks and machines cannot “socialize” (in the strict sense), combining tasks together and machines together presents some similarities with how people behave daily. Different initiatives already demonstrate the successful blend of social software with many disciplines such as learning [6], healthcare [7], and commerce [8].

Tan et al. state that “Currently, most social networks connect people or groups who expose similar interests or features. In the near future, we expect that such networks will connect other entities, such as software components, Web-based services, data resources, and workflows. More importantly, the interactions among people and nonhuman artifacts have significantly enhanced data scientists’ productivity” [9].

Our contributions in this paper include

1. Definition and specification of a set of social (execution as well) relations that permit to connect tasks together, persons together, and machines together in a business process;

2. Development of a set of networks built upon social relations; these networks are referred to as configuration for tasks, support for machines, and social for persons;

3. Demonstration through a proof-of-concept of configuration, support, and social networks development.

4. Brief discussion of the role of configuration, support, and social networks in addressing some issues that hinder business process execution.

The remainder of this paper is organized as follows. Section 2 is an overview of social software. A case study is presented in Section 3. Section 4 introduces the approach to manage social business processes. Prior to concluding in Section 6, a prototype system is discussed in Section 5.

## 2. Overview of social software

We report on what social software refers to and then, on some initiatives that blend social software with business processes. Dustdar and Bhattacharya stress out “the huge gap between business process management technologies, usage patterns, and workflows on the one hand, and social computing as it is known today” [10].

In the literature there is not a common definition of social software. Warr states that “social software includes a large number of tools used for online communication, e.g., instant messaging, text chat, internet fora, weblogs, Wikis, social network services, social guides, social bookmarking, social citations, social libraries, and virtual worlds” [11]. For Schmidt and Nurcan, social software supports productivity by raising the level and scope of interactions because of the use of computers and networks [12]. Erol et al. note that the roots of social software can be traced back to the 40s and add that “impressive results are created without a central plan or organization. Instead, social software uses a self-organization and bottom-up approach where interaction is coordinated by the “collective intelligence” of the individuals; the latter does not necessarily know each other and are a priori not organized in a hierarchy. Furthermore, social software follows a rather egalitarian approach; decisions are not made by small elites but by combining a multitude of inputs from different users” [13]. For Lipitsky et al., social software “fosters collaboration of individuals who work across time, space, cultural, and organizational boundaries” [14]. People engage in conversations and transactions so that common deliverables are produced promptly and with minimum of conflicts. Finally, Bruno et al. identify the four characteristics of social software [15]: (i) weak ties are spontaneously established contacts creating new views on problems and allowing competency combination, (ii) social production breaks with the paradigm of centralized a-priori planning of production and promotes unforeseen and innovative contributors and contributions, (iii) egalitarianism abolishes hierarchical structures, merges the roles of contributors and consumers, and introduces a culture of trust, and (iv) mutual service provisioning changes the cooperation model from a client-server model to a model based on exchanging services.

The blend of social software with business processes is reported throughout the literature. In [16], Rito Silva et al. describe the AGILIPO project that embeds social features into business process tools. The AGILIPO modeling and execution environment includes three roles known as executor, modeler, and developer that stakeholders take over. Executor carries out business processes either by making use of specified activities or by creating generic activities whenever the specified activities are not sufficient. Modeler changes
the business process model by specifying new non-automated activities. Finally developer may consider automating the non-automated activities. To foster collaboration between these stakeholders, social software features such as tagging, versioning, comments, and rating are adopted. In [17], Brambilla et al. propose a specific notation to design social business processes. Social networking helps organizations harness the value of information relations and weak ties without compromising the consolidated business practices that are found in conventional business process management solutions. Despite these benefits there is a lack of appropriate notations that can be used to reflect social aspects on business process models. Brambilla et al.’s notation includes event and task types like broadcast, posting, and invitation to activity. In [18], Koschmider et al. demonstrate how social networks help enhance trust between users. Two networks are built upon a set of business processes and recommendations. The first network provides an organizational view of business processes by suggesting for instance, the average distance between performers who participated in existing business processes and those who are now participating in developing business processes. The second network shows the relations among modelers who use the recommendation system to build the business process model. In [19], Grim-Yefsah et al. reveal the existence of informal networks that people at work rely on to conduct their business. These networks perfectly co-exist with regular networks where formal relations like supervision are reported. Grim-Yefsah et al. discuss how the “official” executor of a task informally seeks other persons’ help in the organization known as contributors. The help takes different forms like asking for advices or confirming technical details. The informal networks back the work of regular networks and do not compete with them.

The aforementioned initiatives on blending social software with business processes develop different solutions such as tagging business processes, using social networks to enhance trust, and mixing formal and informal networks. However what social relations connect a business process’s components, how business processes are adjusted in response to these relations, and what benefits these relations offer to companies, are left unanswered. In this paper we address the first question by showing how to connect tasks together, machines together, and persons together using social relations.

3. Case study

Our case study refers to the electronic-patient-folder system at Anderson Hospital that handles approximately 6000 annual inpatient admissions2. We leverage this system to identify first, some business processes’ components (i.e., tasks, persons and machines) and second, the execution nature of some tasks. When a patient shows up at the hospital, the necessary documentation is scanned into a system known as ImageNow. Upon completion the patient’s MEDITECH record is updated automatically. An advantage of this update is that different stakeholders like billing staff, coders, and other authorized people have immediate, electronic access to the necessary information instead of waiting for the paper documentation to arrive. Prior to implementing the new system Anderson Hospital faced different challenges such as paper records limit access to one user at a time and paper and manual processes hamper compliance with some healthcare standards.

We use this case study to shed the light on the social dimension of business processes by establishing relations between tasks, between machines, and between persons. It is common that events “disturb” the normal completion of business processes due to call-in-sick doctors, last-minute changes in surgery dates, appointment system failure, etc. When dealing with these events we would like to assist the hospital’s managers in considering doctors’ recommendations when looking for substitutes, interchanging tasks to avoid policy violation, and identifying machines based on their coupling level. These are some of the benefits that networks build upon relations between tasks, between machines, and between persons will provide to these managers.

4. Our approach to social business processes

We recall that a business process is “a set of logically related tasks performed to achieve a defined business outcome” [20]. We also recall that the execution of some processes is strictly confined into the borders of single units (e.g., finance department), while the execution of others crosses several independent units raising security, privacy, heterogeneity, and monitoring concerns among IT practitioners and end-users as well [21].

4.1. Overview

Our approach fosters the intertwining of the three components of a business process that are task, person, and machine (Fig. 1). The success of this intertwining depends on (i) identifying execution and

2http://www.perceptivesoftware.com/pdfs/casestudies/psi_cs_anderson.pdf
social relations between tasks (t), between executors (i.e., persons (p) and machines (m)), and between tasks and executors; and (ii) developing categories of networks upon these relations as per the characteristics of each component. These two points are thoroughly discussed in Sections 4.2 and 4.3, respectively.

Fig. 1 illustrates an example of business process for Anderson Hospital case-study. It includes multiple tasks such as t1: scan documentation, t2: update records, and t3: prepare bill. Tasks connect to each other through input and output dependencies, e.g., patient’s data from t1 are sent to t2 so that patient records are updated. However these dependencies are primarily meant for data exchange and thus, do not help much in enriching a business process with any social element nor in shedding the light on the potential social relations between this process’s components. Fig. 1 also shows the execution nature of tasks. Some tasks are completely manual (p1: cashier executing t1) while others are either completely automated (m2: ImageNow executing t2) or semi-automated/semi-manual (p1/m1: operator(scanner/scan taking turns in executing t1).

In preparation for exposing the social dimension of business processes, we associate task with requirements (e.g., t2: update records must be done within one hour of scan receipt), person with capacities (e.g., p1: operate scanner), and machine with capacities as well (e.g., m1: produce high-resolution scan). Requirements impose restrictions on those who will execute tasks in terms of execution nature (e.g., manual), necessary expertise level for persons, reliability level for machines, simultaneous involvement of persons and machines, etc. In addition to requirements we label task as self-contained when its output does not require any additional processing by another task. Additional elements that make a task self-contained are discussed in [22]. Encrypt data is an example of not-self-contained task since decrypting data for proper use is required at a later stage. Unless stated a task is by default self-contained. Task assignment to executors depends on matching requirements to capacities. However the matching does not fall into this paper’s scope.

4.2. Relation identification

We identify relations between tasks, between persons, and between machines from two perspectives: Execution (E) and Social (S). As stated earlier making tasks and machines “socialize” is backed by first, the relations between tasks and between machines that map perfectly onto relations between persons and second, Roush’s statement that computing means connecting [23]. We exemplify the proposed social and execution relations by the case study. It is worth noting that some relations between persons are appropriate for machines and vice-versa. This is primarily due to the semantics of these relations.

Relations between tasks. From an E perspective, execution relations (aka dependencies) between tasks are well defined in the literature [24] such as, prerequisite (e.g., t1 and t2), parallel prerequisite (e.g., t2 and t4: synchronize patient records), and parallel (e.g., t1 and t3: check pending bills). To deal with not self-contained tasks, we propose completion as an additional execution relation between tasks (e.g., compress scan prior to archiving and then decompress scan upon request); t1 and t3 engage in a completion relation when t1 is not self-contained and needs t3 to process its output (e.g., compress and decompress).

From a S perspective, we propose two social relations:

- **Interchange**: t1 and t2 engage in an interchange relation when both produce similar output with respect to similar input received for processing and their requirements do not overlap (e.g., t1 and t2: enter patient details manually in the case the scanner is down). The non-overlap condition is necessary to avoid blockage when t1’s requirements (e.g., online data entry) cannot be met due to lack of executors and thus, t1 needs to be interchanged with t2 that has different requirements (e.g., offline data entry). In terms of benefits, interchange indicates how difficult a task’s requirements are satisfied if the task is constantly replaced and what tasks are frequently used as replacements.

- **Coupling**: t1 and t2 engage in a coupling relation when they interact in the same business processes through one of the aforementioned execution relations (excluding completion). In terms of benefits, coupling indicates how strong or weak the connection between tasks is, which should help recommend tasks when putting business processes together at design time.

Relations between machines. In companies, machines (e.g., scanner and ImageNow) ensure the performance of automated and semi-automated tasks. Each machine (m1) is overseen by a dedicated software component for management purposes, but this is outside this paper’s scope. As stated earlier, matching capacities to requirements identifies the necessary machines that will execute tasks.

From an E perspective, we identify execution relations between m1 and m2 by analyzing Decker and Lesser’s six coordination relations between tasks [25]. These relations are enables, facilitates, cancels, constrains, inhibits, and causes, and only two are considered as per our needs of connecting machines together to execute joint tasks: (i) enablement is established when m1 produces (internal) output that allows m2 to continue
the execution of the task (e.g., \( t_3: \text{do chest Xray requiring} \ m_x: \text{Xray machine to take Xrays and} \ m_{t3}: \text{communication app. to send the doctor Xrays} \), and (ii) inhibition is established when \( m_j \) executes its part of the task after a certain time elapses since \( m_i \) completed the execution of its part (e.g., \( t_2: \text{do blood test where the results are sent by email some time later after blood collection and analysis using specialized equipment} \)). It is worth mentioning that defining execution relations between independent machines in charge of dependent tasks is not relevant for our work. The previous execution relations between tasks in the \( E \)-perspective impose automatically an execution chronology on the machines.

From a \( S \) perspective, we consider three social relations:

- **Backup**: \( m_i \) (e.g., scanner in main reception) and \( m_j \) (e.g., 3-in-1 printer in nurse station) engage in a backup relation when both have similar capacities (subsumption relation between capacities can also be considered using ontology). In terms of benefits, backup indicates how reliable a machine is and what machines are frequently requested as backups.

- **Cooperation**: \( m_i \) and \( m_j \) engage in a cooperation relation when both are already engaged in a backup relation (because of similar capacities) and the simultaneous combination of their respective capacities is necessary to meet a task’s requirements. In terms of benefits, cooperation indicates how often similar machines work together on executing tasks, which should help recommend machines when carrying out the matching. Though the machines are similar their collective performance might not correspond to the combination of their individual performances.

- **Partnership**: \( m_i \) and \( m_j \) engage in a partnership relation when their capacities are complementary and the simultaneous combination of their respective capacities is necessary to meet a task’s requirements (e.g., \( t_3: \text{analyze blood sample requiring different types of machines} \)). In terms of benefits, partnership indicates how often separate machines work together on tasks, which should help recommend machines when carrying out the matching. Since machines are different their collective performance needs to take into account the particularities of each in terms of individual performances and functional constraints, for example.

**Relations between persons.** In companies persons make decisions, trigger processes, exchange information, etc. As stated earlier, matching capacities to requirements identifies the necessary persons to execute tasks.

From an \( E \) perspective, we adopt the same execution relations for machines in order to connect \( p_i \) and \( p_j \) that are assigned joint tasks: (i) enablement is established when \( p_i \) performs other (internal) tasks whose completion allows \( p_j \) to continue the execution of the task (e.g., \( t_3: \text{check patient requiring} \ m_{t3}: \text{nurse to check patient’s vitals and} \ m_{p3}: \text{doctor to consult patient and provide medication} \)), and (ii) inhibition is established when \( p_j \) executes her part of the task after a certain time elapses since \( p_i \) completed the execution of her part (e.g., \( t_2: \text{perform surgery requiring} \ m_{t2}: \text{anesthetist and} \ m_{p2}: \text{surgeon who intervenes after the anesthetic becomes effective} \)).

From a \( S \) perspective, we consider three social relations:

- **Substitution**: \( p_i \) (e.g., general practitioner in family medicine) and \( p_j \) (emergency physician in emergency department) engage in a substitution relation when both have similar capacities (subsumption relation between capacities can also be considered using ontology). In terms of benefits, substitution indicates how available a person is and what persons are frequent substitutes.

- **Delegation**: \( p_i \) and \( p_j \) engage in a delegation relation when both are already engaged in a substitution relation (in term of capacity assessment) and \( p_j \) decides to assign a task that she will execute or is now executing to \( p_j \) due to unexpected changes in her status, e.g., call-in-sick or situation of overload (e.g., emergency
physician transfers patient to general practitioner due to the arrival of an urgent case). In terms of benefits, delegation indicates the transfer-of-work between persons, which could help identify the right persons next time unexpected events arise.

- **Peering:** $p_i$ and $p_j$ engage in a peering relation when both are in different organizational units and their respective similar (e.g., managerial) and complementary (e.g., expertise) capacities are necessary to meet a task’s requirements. In terms of benefits, peering fosters cross-organization activities and indicates how often persons work together on common tasks, which should help recommend persons when the matching of capacities to requirements occurs.

**Table 1.** Substitution versus delegation

<table>
<thead>
<tr>
<th>Substitution($p_i$,$p_j$)</th>
<th>Delegation($p_i$,$p_j$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_i$ stops executing ongoing tasks; $p_j$ executes these tasks</td>
<td>$p_i$ continues executing other ongoing tasks; $p_j$ executes tasks assigned by $p_i$</td>
</tr>
<tr>
<td>$p_j$ reports to whom $p_i$ reports upon task completion; otherwise $p_j$ waits for $p_i$ return</td>
<td>$p_j$ reports to $p_i$ upon task completion</td>
</tr>
</tbody>
</table>

Table 1 compares substitution to delegation in terms of who does what and who reports to whom. Table 2 summarizes the social relations between tasks, between machines, and between persons along with their respective pre-conditions, conditions, and post-conditions. Pre-condition defines the rationale of a social relation between a process’s components. Condition indicates when a network built upon a social relation is used so that solutions to conflicts that prevent a business process completion are addressed. Finally post-condition indicates the successful resolution of these conflicts so that network use is stopped. In Table 2 the occurrence of multiple relations (e.g., backup and cooperation) between the components of the same process is not exclusive if these relations’ pre-conditions are simultaneously satisfied.

### 4.3. Network categorization

The social relations presented earlier are used for developing specialized networks, i.e., one network per relation. We group these networks into three categories: configuration network of tasks, support network of machines, and social network of persons. In this part of the paper we analyze the nodes and edges per network category and evaluate the edges that connect nodes.

**Configuration network of tasks.** In a configuration network of tasks, node and edge correspond to task and relation between tasks, respectively. We analyze both task and relation from $\mathcal{E}$ and $\mathcal{S}$ perspectives. On the one hand the $\mathcal{E}$ perspective sets the stage for connecting tasks together in order to form business processes and assigning tasks to executors. On the other hand the $\mathcal{S}$ perspective sets the stage for building networks of tasks based on Table 2’s relations (pre-condition satisfaction) as well as using these networks when necessary (condition satisfaction).

**Task.** In the $\mathcal{E}$ perspective, a task ($t_i$) is a concrete work unit (e.g., do blood test) that acts upon the environment. We define $t_i$ as a 5-tuple $\langle \mathcal{E}$-Req$_i, \mathcal{E}$-Pre-Condition$_i, \mathcal{E}$-Input/Output$_i, \mathcal{E}$-Condition$_i, \mathcal{E}$-Post-Condition$_i \rangle$ where $\mathcal{E}$-Req$_i$ is $t_i$’s requirements, $\mathcal{E}$-Pre-Condition$_i$ is a set of physical and logical elements to verify so that $t_i$ is assigned to an executor (i.e., $\mathcal{E}$-Req$_i$ is satisfied with respect to the executor’s capacities), $\mathcal{E}$-Input$_{t_i}$ and $\mathcal{E}$-Output$_{t_i}$ identify, respectively, the data that $t_i$ may need for execution and the data that $t_i$ may produce after execution, $\mathcal{E}$-Condition$_{t_i}$ is a set of physical and logical elements to verify so that $t_i$ can be executed after successful assignment to an executor (e.g., all necessary inputs are available), and $\mathcal{E}$-Post-Condition$_{t_i}$ is a set of physical and logical elements to verify so that the execution of $t_i$ is declared either successful or failure. In the case of failure corrective actions are taken and vary from one case study to another.

In the $\mathcal{S}$ perspective, a task ($t_i$) is an abstract work unit (i.e., narrative description) that “signs up” in the interchange and/or coupling networks. We define $t_i$ as a couple $\langle \mathcal{S}$-Network$_{rel}, \mathcal{S}$-Connect$_{rel}(t_i,w_j)\rangle$ where $\mathcal{S}$-Network$_{rel}$ is either interchange network or coupling network, $\mathcal{S}$-Connect$_{rel}(t_i,w_j)$ lists all $t_j=1..n, j \neq i \in \mathcal{S}$-Network$_{rel}$ that are connected to $t_i$ through $rel$, and $w_j$ is the weight of the edge from $t_i$ to $t_j$. Weight assessment is given a little bit later.

**Relation.** In the $\mathcal{E}$ perspective, a relation ($r_{(t_i,t_j)}$) establishes a concrete dependency between $t_i$ and $t_j$ at run time. A dependency can be either direct (i.e., $t_i$ follows $t_j$) or indirect (i.e., because $t_j$ is not self-contained, $t_j$ is called whenever $t_i$’s output needs to be processed). To handle task dependencies we decompose Input$_{t_j}$ into either initialInput$_{t_j}$ (i.e., submitted to $t_j$ prior to execution) or partialInput$_{t_j}$ (submitted to $t_j$ during execution) and also decompose Output$_{t_i}$ into either partialOutput$_{t_i}$ (i.e., produced by $t_i$
Table 2. Summary of social relations

<table>
<thead>
<tr>
<th>Between Types</th>
<th>Pre-Conditions</th>
<th>Conditions</th>
<th>Post-Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_i, t_j )</td>
<td>Coupling</td>
<td>( t_i ) and ( t_j ) participated in joint business processes</td>
<td>review of business process design or concern over coupling level</td>
</tr>
<tr>
<td></td>
<td>Interchange</td>
<td>( t_i ) and ( t_j ) producing similar output in receipt of similar input</td>
<td>( t_i ) lacking of executor who satisfies its requirements</td>
</tr>
<tr>
<td>( m_i, m_j )</td>
<td>Backup</td>
<td>( m_i ) and ( m_j ) having similar capacities</td>
<td>( m_i ) unexpected failure or concern over ( m_i ) reliability</td>
</tr>
<tr>
<td></td>
<td>Cooperation</td>
<td>( m_i ) and ( m_j ) having similar capacities</td>
<td>concern over machine collective performance</td>
</tr>
<tr>
<td></td>
<td>Partnership</td>
<td>( m_i ) and ( m_j ) having complementary capacities</td>
<td>concern over machine collective performance</td>
</tr>
<tr>
<td>( p_i, p_j )</td>
<td>Substitution(^1)</td>
<td>( p_i ) and ( p_j ) having similar capacities</td>
<td>( p_i ) expected unavailability (e.g., annual leave and sick leave) or concern over ( p_i ) availability</td>
</tr>
<tr>
<td></td>
<td>Delegation</td>
<td>( p_i ) and ( p_j ) having similar capacities</td>
<td>( p_i ) unexpected unavailability (e.g., call-in-sick, urgent tasks to complete, and risk of overload)</td>
</tr>
<tr>
<td></td>
<td>Peering</td>
<td>( p_i ) and ( p_j ) having similar or complementary capacities</td>
<td>concern over peering appropriateness</td>
</tr>
</tbody>
</table>

\(^1\) Substitution example: \((t_1: \text{scan documentation})\) and \((t'_1: \text{enter patient details manually})\) are connected together in the substitution network since the interchange pre-condition is met, i.e., producing similar output (e.g., patient’s healthcare provider) in receipt of similar input (e.g., patient’s number). This network will be used when the interchange condition is met, which is scanner down so \( t_1 \) cannot be executed and replaced with \( t'_1 \). After identifying an executor for \( t'_1 \) who is the agent at the reception, the use of this network is stopped since the interchange pre-condition is met.
during execution) or finalOutputi (i.e., produced by t at the end of execution). Unless stated Input and Output refer to initialInput and finalInput, respectively. We define \( r_{(t_i,t_j)} \) as a triple \(<t_i, t_j, \mathcal{E}\text{-Type}>\) where \( \mathcal{E}\text{-Type} \) is either prerequisite dependency (i.e., \( t_i \) successful execution as per \( \mathcal{E}\text{-Pre-Condition}_{t_i} \) and \( \mathcal{E}\text{-finalOutput}_{t_i} \cap \mathcal{E}\text{-initialInput}_{t_i} \neq \emptyset \)), parallel-prerequisite dependency (i.e., \( \mathcal{E}\text{-partialOutput}_{t_i} \cap \mathcal{E}\text{-initialInput}_{t_i} \neq \emptyset \)), parallel dependency (\( \mathcal{E}\text{-partialOutput}_{t_i} \cap \mathcal{E}\text{-finalOutput}_{t_i} = \emptyset \), and \( \mathcal{E}\text{-finalOutput}_{t_i} \cap \mathcal{E}\text{-partialInput}_{t_i} \neq \emptyset \), or completion relation (\( \mathcal{E}\text{-Output}_{t_i} \) requires processing).

In the \( S \) perspective a relation \( r_{(t_i,t_j)} \) establishes a link between \( t_i \) and \( t_j \) both members of a network that is built upon this relation. We define \( r_{(t_i,t_j)} \) as a 5-tuple \(<t_i, t_j, \mathcal{S}\text{-Pre-Condition}_{(t_i,t_j)}, \mathcal{S}\text{-Condition}_{(t_i,t_j)}, \mathcal{S}\text{-Post-Condition}_{(t_i,t_j)}>\) where \( \mathcal{S}\text{-Pre-Condition}_{(t_i,t_j)} \) is a set of physical and logical elements to verify so that \( t_i \) connects \( t_j \) (i.e., either \( \mathcal{E}\text{-Output}_{t_i} \) and \( \mathcal{E}\text{-Output}_{t_j} \) are equivalent with respect to similar input received) or \( (t_i \) and \( t_j \) participate in joint business processes)), \( \mathcal{S}\text{-Condition}_{(t_i,t_j)} \) is a set of physical and logical elements to verify so that the network associated with \( r_{(t_i,t_j)} \) is used, and \( \mathcal{S}\text{-Post-Condition}_{(t_i,t_j)} \) is a set of physical and logical elements to verify before the ongoing use of the network associated with \( r_{(t_i,t_j)} \) is stopped (i.e., executor found for \( t_j \), business process enrichment completion, or coupling level satisfaction).

**Relation evaluation.** Assessing the weight \((w)\) of \( r_{(t_i,t_j)} \) is restricted to the \( S \) perspective, only.

- **Interchange relation.** Equation 1 measures the weight of an interchange edge where \([\text{interchange}_{(t_i,t_j)}]\) is the number of times that \( t_i \) is interchanged successfully with \( t_j \) (i.e., an executor is found for \( t_j \)) and \([\text{failure}_{t_i}]\) is the number of times that \( t_i \) is not executed due to lack of executors. A higher combined interchange weight for a task (e.g., close to 1) indicates the excessive interchange of this task, which should be taken into account when implementing critical business processes.

\[
w_{\text{interchange}_{(t_i,t_j)}}^S = \frac{[\text{interchange}_{(t_i,t_j)}]}{[\text{failure}_{t_i}]} \tag{1}\]

- **Coupling relation.** Equation 2 measures the weight of a coupling edge where \([\text{participateJointProcess}_{(t_i,t_j)}]\) is the number of times that \( t_i \) is directly coupled with \( t_j \) in joint business processes and \([\text{participateProcess}_{t_i}]\) is the number of times that \( t_i \) participates in business processes. A higher coupling weight (e.g., close to 1) indicates the “smoothness” of exchanging data between tasks due to “limited” semantic mismatches.

\[
w_{\text{coupling}_{(t_i,t_j)}}^S = \frac{[\text{participateJointProcess}_{(t_i,t_j)}]}{[\text{participateProcess}_{t_i}]} \tag{2}\]

**Support network of machines.** In a support network of machines, node and edge correspond to machine and relation between machines, respectively. We analyze both from \( \mathcal{E} \) and \( S \) perspectives. On the one hand the \( \mathcal{E} \) perspective sets the stage for assisting machines schedule task execution and also initiating this execution. On the other hand the \( S \) perspective sets the stage for building networks of machines using the relations listed in Table 2 (pre-condition satisfaction) and also using these networks when needed (condition satisfaction).

**Machine.** In the \( \mathcal{E} \) perspective, a machine \((m_i)\) is a concrete processing unit (e.g., printer) that receives tasks (e.g., print out document) for execution after confirming that the machine’s capacities meet these tasks’ requirements, i.e., tasks’ pre-conditions satisfied. Like with task we define \( m_i \) as a 5-tuple \(<\mathcal{E}\text{-Cap}_{m_i}, \mathcal{E}\text{-Pre-Condition}_{m_i}, \mathcal{E}\text{-Input/Output}_{m_i}, \mathcal{E}\text{-Condition}_{m_i}, \mathcal{E}\text{-Post-Condition}_{m_i}>\) where \( \mathcal{E}\text{-Cap}_{m_i} \) is \( m_i \)'s capacities, \( \mathcal{E}\text{-Pre-Condition}_{m_i} \) is a set of physical and logical elements to verify so that \( m_i \) schedules the execution of a task taking into account \( \mathcal{E}\text{-Cap}_{m_i} \), \( \mathcal{E}\text{-Input}_{m_i} \) and \( \mathcal{E}\text{-Output}_{m_i} \) (optional) identify, respectively, the tasks that \( m_i \) receives for execution and the results (e.g., data and product) that \( m_i \) may produce after execution, \( \mathcal{E}\text{-Condition}_{m_i} \) is a set of physical and logical elements to verify so that \( m_i \) begins executing a task (e.g., no pending tasks exist), and \( \mathcal{E}\text{-Post-Condition}_{m_i} \) is a set of physical and logical elements to verify so that \( m_i \) detaches a task after execution and updates \( \mathcal{E}\text{-Cap}_{m_i} \). The success or failure of this execution is dependent on the task’s post-conditions.

In the \( S \) perspective, a machine \((m_i)\) is an abstract processing unit (i.e., narrative description) that “signs up” in networks built upon the three
possible social relations between machines. We define $m_i$ as a couple $[\text{S-Network}_{rel}, \text{1} \text{S-Connect}_{rel}(m_j, w_j)]$$ \ni \text{T}_{m_i}$ where $\text{S-Network}_{rel}$ is either backup network, cooperation network, or partnership network, $\text{S-Connect}_{rel}(m_j, w_j)$ lists all $m_j \ni \text{T}_{m_i}$ connected to $m_i$ through $rel$, and $w_j$ is the weight of the edge from $m_i$ to $m_j$. Weight assessment is given a little bit later.

**Relation.** In the $\mathcal{E}$ perspective, a relation ($r_{(m_i, m_j)}$) establishes a link between two machines $m_i$ and $m_j$ both in charge of executing joint tasks. We define $r_{(m_i, m_j)}$ as a triple $<m_i, m_j, \mathcal{E}\text{-Type}>$ where $\mathcal{E}\text{-Type}$ is either enablement relation or inhibition relation.

In the $\mathcal{S}$ perspective, a relation ($r_{(m_i, m_j)}$) establishes a link between two machines $m_i$ and $m_j$ both members of a network built upon this relation. We define $r_{(m_i, m_j)}$ as a 5-tuple $<m_i, m_j, \mathcal{S}\text{-Pre-Condition}_{m_i, m_j}, \mathcal{S}\text{-Condition}_{m_i, m_j}, \mathcal{S}\text{-Post-Condition}_{m_i, m_j}>$ where $\mathcal{S}\text{-Pre-Condition}_{m_i, m_j}$ is a set of physical and logical elements to check so that the network associated with $r_{(m_i, m_j)}$ is used (i.e., either $m_i$ fails unexpectedly, concern over $m_i$ reliability, or concern over the collective performance of $m_i$ and $m_j$), and $\mathcal{S}\text{-Post-Condition}_{m_i, m_j}$ is a set of physical and logical elements to check before the ongoing use of the network associated with $r_{(m_i, m_j)}$ is stopped (i.e., backup/replacement machine found for $m_i$ or satisfaction with collective performance of $m_i$ and $m_j$).

**Relation evaluation.** Assessing the weight ($w$) of $r_{(m_i, m_j)}$ is restricted to the $\mathcal{S}$ perspective, only. In the following $\mathcal{T}_{m_i}$ is the set of all tasks assigned to $m_i$.

- **Backup relation.** Equation 3 measures the weight of a backup edge where $\mathcal{T}_{m_i}^{\text{fail}} \subseteq \mathcal{T}_{m_i}$ is the set of tasks that $m_i$ failed to execute and $|\text{replaceSuc}_{m_i}^{\text{fail}}(m_j, m_i)|$ is the number of failed tasks in $\mathcal{T}_{m_i}^{\text{fail}}$ that $m_j$ completes successfully. A higher combined backup weight for a machine (e.g., close to 1) indicates its reliability, which should be taken into account prior to finalizing task assignments.

$$w_{\text{backup}}^{\mathcal{S}}(m_i, m_j) = \frac{|\text{replaceSuc}_{m_i}^{\text{fail}}(m_j, m_i)|}{|\mathcal{T}_{m_i}^{\text{fail}}|}$$ (3)

- **Cooperation relation.** Equation 4 measures the weight of a cooperation edge where $\mathcal{T}_{m_i}^{\text{coop}} \subseteq \mathcal{T}_{m_i}$ is the set of all tasks that $m_i$ executes in cooperation with other machines and $|\text{cooperationSuc}_{m_i}^{\text{coop}}(m_j, m_i)|$ is the number of tasks in $\mathcal{T}_{m_i}^{\text{coop}} \cap \mathcal{T}_{m_j}^{\text{coop}}$ that $m_i$ and $m_j$ execute successfully together. A cooperation weight reveals the collective performance of similar machines executing joint tasks.

$$w_{\text{cooperation}}^{\mathcal{S}}(m_i, m_j) = \frac{|\text{cooperationSuc}_{m_i}^{\text{coop}}(m_j, m_i)|}{|\mathcal{T}_{m_i}^{\text{coop}}|}$$ (4)

- **Partnership relation.** Equation 5 measures the weight of a partnership edge where $\mathcal{T}_{m_i}^{\text{part}} \subseteq \mathcal{T}_{m_i}$ is the set of all tasks that $m_i$ executes in partnership with other machines and $|\text{partnerSuc}_{m_i}^{\text{part}}(m_j, m_i)|$ is the number of tasks in $\mathcal{T}_{m_i}^{\text{part}} \cap \mathcal{T}_{m_j}^{\text{part}}$ that $m_i$ and $m_j$ complete successfully together. A partnership weight reveals the collective performance of different machines executing joint tasks.

$$w_{\text{partnership}}^{\mathcal{S}}(m_i, m_j) = \frac{|\text{partnerSuc}_{m_i}^{\text{part}}(m_j, m_i)|}{|\mathcal{T}_{m_i}^{\text{part}}|}$$ (5)

**Social networks of persons.** In a social network of persons, node and edge correspond to person and relation between persons, respectively. We analyze both from $\mathcal{E}$ and $\mathcal{S}$ perspectives. On the one hand the $\mathcal{E}$ perspective sets the stage for assisting persons plan task execution and also initiating this execution. On the other hand the $\mathcal{S}$ perspective sets the stage for building networks of persons using the relations listed in Table 2 (pre-condition satisfaction) and also using these networks when needed (condition satisfaction).

- **Person.** In the $\mathcal{E}$ perspective, a person ($p_i$) is a concrete “processing unit” (e.g., doctor) who receives tasks (e.g., check patient) for execution after confirming that the person's capacities meet these tasks' requirements, i.e., tasks' pre-conditions satisfied. Like with tasks and machines we define $p_i$ as a 5-tuple $<\mathcal{E}\text{-Cap}_{p_i}, \mathcal{E}\text{-Pre-Condition}_{p_i}, \mathcal{E}\text{-Input/Output}_{p_i}, \mathcal{E}\text{-Condition}_{p_i}, \mathcal{E}\text{-Post-Condition}_{p_i}>$ where $\mathcal{E}\text{-Cap}_{p_i}$ is $p_i$'s capacities, $\mathcal{E}\text{-Pre-Condition}_{p_i}$ is a set of physical and logical...
elements to verify so that \( p_i \) plans the execution of a task taking into account \( \mathcal{E} \)-Cap\(_{p_i} \), \( \mathcal{E} \)-Input\(_{p_i} \), and \( \mathcal{E} \)-Output\(_{p_j} \) identify, respectively, the tasks that \( p_i \) receives for execution and the tasks/results that \( p_i \) might generate/produce after execution, \( \mathcal{E} \)\-Condition\(_{p_i} \) is a set of physical and logical elements to verify so that \( p_i \) begins executing a task (e.g., no-higher priority tasks exist), and \( \mathcal{E} \)\-Post-Condition\(_{p_i} \) is a set of physical and logical elements to verify so that \( p_i \) detaches a task after execution and updates \( \mathcal{E} \)-Cap\(_{p_i} \). The success or failure of this execution is dependent on the task’s post-conditions.

In the \( S \) perspective, a person (\( p_i \)) is an abstract “processing unit” (i.e., narrative description) who signs up in networks built upon the three possible relations between persons namely, substitution, delegation, or peering. We define \( p_i \) as a couple \(|S\)-Network\(_{rel} \), \(|S\)-Connect\(_{rel}(p_i, w_j)\)|\( \subseteq \mathcal{E} \) where \(|S\)-Network\(_{rel} \) is either substitution network, delegation network, or peering network, \(|S\)-Connect\(_{rel}(p_i, w_j)\)| lists all \( p_j=1...n \) connected to \( p_i \) through \( rel \), and \( w_j \) is the weight of the edge from \( p_i \) to \( p_j \). Weight assessment is given a little bit later.

- **Relation**. In the \( \mathcal{E} \) perspective, a relation \( \langle r(p_i, p_j) \rangle \) establishes a link between two persons \( p_i \) and \( p_j \) in charge of executing joint tasks. We define \( r(p_i, p_j) \) as a triple \( \langle m_i, m_j, \mathcal{E} \rangle \) where \( \mathcal{E} \)\-Type is either enablement relation or inhibition relation.

In the \( S \) perspective, a relation \( \langle r(p_i, p_j) \rangle \) establishes a link between two persons \( p_i \) and \( p_j \) both members of a network built upon this relation. We define \( r(p_i, p_j) \) as a 5-tuple \( \langle p_i, p_j, S\)-Pre-Condition\(_{p_i, p_j} \), \( S\)-Condition\(_{p_i, p_j} \), \( S\)-Post-Condition\(_{p_i, p_j} \rangle \) where \( S\)-Pre-Condition\(_{p_i, p_j} \), \( S\)-Condition\(_{p_i, p_j} \), and \( S\)-Post-Condition\(_{p_i, p_j} \) is a set of physical and logical elements to check so that \( p_i \) connects \( p_j \) (i.e., \( \mathcal{E} \)-Cap\(_{p_i} \) and \( \mathcal{E} \)-Cap\(_{p_j} \) are similar or complementary), \( S\)-Condition\(_{p_i, p_j} \) is a set of physical and logical elements to verify so that the network associated with \( r(p_i, p_j) \) is used (i.e., \( p_i \) is unavailable, concern over \( p_i \) or \( p_j \) availability, or concern over peering \( p_i \) and \( p_j \) together), and \( S\)-Post-Condition\(_{p_i, p_j} \) is a set of physical and logical elements to check before the ongoing use of the network associated with \( r(p_i, p_j) \) is stopped (i.e., substitute/delegate was identified for \( p_i \) or peer found for either \( p_i \) or \( p_j \)).

- **Relation evaluation**. Assessing the weight \( (w) \) of \( r(p_i, p_j) \) is restricted to the \( S \) perspective, only. In the following \( T_p \) is the set of tasks assigned to \( p_i \).

- **Substitution relation**. Equation 6 measures the weight of a substitution edge where \( T_{p_i} \subseteq T_{p_j} \), the set of all tasks assigned to \( p_i \), but then are assigned to a different person due to \( p_i \) expected unavailability and \( |\text{substituteSuc}_{T_{p_i}}(p_i, p_j)| \) is the number of tasks in \( T_{p_i}^{\text{sub}} \) that \( p_j \) completes successfully.

\[
w_{\text{substitution}(p_i, p_j)} = \frac{|\text{substituteSuc}_{T_{p_i}}(p_i, p_j)|}{|T_{p_i}^{\text{sub}}|}
\]

- **Delegation relation**. Equation 7 measures the weight of a delegation edge where \( T_{p_i}^{\text{del}} \subseteq T_{p_i} \) is the set of all tasks assigned to \( p_i \) but then are delegated to \( p_j \) due to her unexpected unavailability and \( |\text{delegateSuc}_{T_{p_i}}(p_i, p_j)| \) is the number of tasks in \( T_{p_i}^{\text{del}} \) that \( p_j \) delegates.

\[
w_{\text{delegation}(p_i, p_j)} = \frac{|\text{delegateSuc}_{T_{p_i}}(p_i, p_j)|}{|T_{p_i}^{\text{del}}|}
\]

- **Peering relation**. Equation 8 measures the weight of a peering edge where \( T_{p_i}^{\text{peer}} \subseteq T_{p_i} \) is the set of all tasks that \( p_i \) executes with other peers and \( |\text{peeringSuc}_{T_{p_i}}(p_i, p_j)| \) is the number of tasks in \( T_{p_i}^{\text{peer}} \) that \( p_i \) and \( p_j \) that both complete successfully together in different business processes. A higher combined peering weight for a person (e.g., close to 1) indicates her work-of-transfer level to other persons.

\[
w_{\text{peering}(p_i, p_j)} = \frac{|\text{peeringSuc}_{T_{p_i}}(p_i, p_j)|}{|T_{p_i}^{\text{peer}}|}
\]

4.4. Role of networks in addressing conflicts

In [26] we detail the role of the networks in addressing conflicts that hinder BP execution. This role refers to BP social coordination that includes four steps: categorize resources that tasks require for their execution, define how tasks/machines/persons of a BP bind to resources during this execution, categorize conflicts on resources that arise between tasks,
between machines, and between persons, and finally analyze the appropriateness of certain networks of tasks/persons/machines for addressing these conflicts.

First we categorize resources into (i) logical, i.e., their use/consumption does not lead into a decrease in their reliability/availability level and (ii) physical, i.e., their use/consumption does lead into a decrease in their reliability/availability level. This decrease requires resource replacement\(^3\)/replenishment at a certain stage. For the sake of illustration in this paper we restrict discussions to logical resources, only. Afterwards we define a set of properties that allow to describe a resource (Table 3). These properties are unlimited (\(ul\), by default), limited (\(l\) - when a resource use/consumption is measured or a resource ceases to exist due to temporal constraints, for example), and limited but renewable (\(lr\) - when a resource use/consumption either hits a certain threshold or is subject to temporal constraints, for example; in either case renewal is possible). Additional properties could be considered if need be, such as non-shareable (\(ns\) - when a resource simultaneous use/consumption has to be scheduled) and shareable (\(s\), by default).

### Table 3. Examples of logical resources per property type

<table>
<thead>
<tr>
<th>Category</th>
<th>Property</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>Unlimited ((ul)) (default)</td>
<td>Data (read mode), software (no cap on number of licences)</td>
</tr>
<tr>
<td></td>
<td>Limited ((l))</td>
<td>Thread</td>
</tr>
<tr>
<td>Limited but</td>
<td>Renewable ((lr))</td>
<td>File access right (valid for a certain time with possible extension), password (valid for a certain time with possible extension)</td>
</tr>
</tbody>
</table>

\(T\)-Conflict\(_1\) is an example of conflicts between tasks with emphasis on resource consumption and not data (inputs and outputs) and policy incompatibilities between these tasks. \(T\)-Conflict\(_1\) arises when (i) a prerequisite relation between \(t_i\) and \(t_j\) exists, (ii) consume\((t_i,t_j)\) \(\rightarrow \) produce\((t_i,t_j)\), and (iii) \(t_j\) needs \(t_i\) (i.e., \(t_j \rightarrow t_i\), no \(t_i\) is made available for \(t_j\)). Potential conflicts on \(t_{ij}\) (and eventually \(t_{ik}::t_{kj}::t_{ij}::t_{kj}\)) become of the limited property of \(t_{ij}\), include:

- \(l\): two cases result out of the prerequisite relation between \(t_{ik}::t_{kj}\) (e.g., complete necessary paperwork) and \(t_j\) (e.g., direct patient to appropriate department) on top of the same relation between \(t_j\) (e.g., check patient vitals) and \(t_i\):
  - \(t_{ij}\) (e.g., report on vital levels) ceases to exist (e.g., blood sample no longer valid) before the execution of \(t_j\) begins; \(t_i\) waits for \(t_{ik}::t_{kj}\) to produce \(t_{ik}::t_{kj}\) (e.g., insurance provider approval); (at least one) \(t_{ik}::t_{kj}\) either is still under execution (e.g., due to delay in receiving approval from insurance provider) or failed.
  - Only one consumption cycle of \(t_{ij}\) is permitted (per type of property) but it turns out that several consumption cycles of \(t_{ij}\) are required to complete the execution of \(t_i\) and finish the consumption of \(t_{ik}::t_{kj}\) that \(t_{ik}::t_{kj}\) produce.

After identifying the different task conflicts on resources, we suggest solutions for these conflicts based on the aforementioned networks. These solutions consider the fact that tasks are associated with transactional properties (e.g., pivot, retrievable, and compensable) that limit their re-execution in the case of failure [27]. The following examines briefly how the interchange and coupling networks of tasks are used to \(T\)-Conflict\(_1\):

- a) \(t_{ij}\) ceases to exist before the execution of \(t_i\) begins; \(t_i\) waits for \(t_{ik}::t_{kj}\) to produce \(t_{ik}::t_{kj}\); at least one \(t_k\) either is still under execution or failed. Current statuses of tasks and resources are: state\((t_j)\): done; state\((t_{ij})\): withdrawn; state\((t_i)\): not-activated; and state\((t_k)\): either activated (still under execution) or failed with focus on the latter state below. Because \(t_j\) now takes on done state, pivot (canceling \(t_j\)) and retrievable (re-executing \(t_j\)) transactional properties are excluded from the analysis of developing solutions to address resource conflicts. This analysis is given in Table 4. The objective is to re-produce \(t_{ij}\) (or produce \(t_{ij}^\prime\) with \(t_j\) being obtained through the interchange network of \(t_{ij}\)). Because of \(t_k\) failure, \(t_{ij}^\prime\) is produced using \(t_k\) that is obtained through the interchange network of \(t_k\).

### 5. Implementation

In this section we describe the architecture of the Social-based Business Process management framework (SUPER) that supports our research on the social enterprise (or enterprise 2.0) [28]. SUPER is a Business Process Management System (BPMS) that uses social computing principles (i.e., connecting entities together through relations, developing networks upon these relations, and analyzing these networks) to model and develop business processes. Fig. 2 illustrates the architecture of SUPER that is built upon multiple components discussed below.

At design time, process engineers (or designers) define new Business Processes (BP)s using the BP modeling component. This component is an extension of the Yaoqiang BPMN editor [29] with new operations that for instance, assign executors (persons and/or machines) to tasks and specify the requirements and capacities of tasks, persons, and machines accordingly. Fig. 3 shows...
<table>
<thead>
<tr>
<th>Transactional property</th>
<th>Coordination actions</th>
<th>Network involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>Null</td>
<td>N/A</td>
</tr>
<tr>
<td>Retriable</td>
<td>- re-execute ( t_i ) to re-produce ( r_{i,j} ) &lt;br&gt; - re-execute ( t_k ) to produce ( u_{k,j} )</td>
<td>N/A</td>
</tr>
<tr>
<td>Compensatable</td>
<td>- compensate ( t_j ); either re-execute ( t_j ) to re-produce ( r_{i,j} ) or replace ( t_j ) with ( t_j' ) then execute ( t_j' ) to produce ( r_{i,j}' ) &lt;br&gt; - either re-execute ( t_k ) to produce ( r_{k,j} ) or replace ( t_k ) with ( t_k' ) then execute ( t_k' ) to produce ( r_{k,j}' )</td>
<td>Interchange(( t_k, t_k' ))</td>
</tr>
<tr>
<td>Compensatable</td>
<td>- compensate ( t_j ); either re-execute ( t_j ) to re-produce ( r_{i,j} ) or replace ( t_j ) with ( t_j' ) then execute ( t_j' ) to produce ( r_{i,j}' ) &lt;br&gt; - replace ( t_k ) with ( t_k' ) then execute ( t_k' ) to produce ( r_{k,j}' )</td>
<td>Interchange(( t_k, t_k' ))</td>
</tr>
<tr>
<td>Retriable</td>
<td>- compensate ( t_j ); either re-execute ( t_j ) to re-produce ( r_{i,j} ) or replace ( t_j ) with ( t_j' ) then execute ( t_j' ) to produce ( r_{i,j}' ) &lt;br&gt; - re-execute ( t_k ) to produce ( u_{k,j} )</td>
<td>Interchange(( t_k, t_k' ))</td>
</tr>
</tbody>
</table>
How to Make Business Processes “Socialize”?

At run time, BPs are executed using appropriate engines for instance, Jboss JBPM engine⁴. This engine interprets the specification of the BP model that is generated from the Jboss JBPM editor. The different execution traces accumulated during the execution of BP instances are stored in a log file. These traces contain information about events referring to the execution of tasks in terms of execution time (timestamp) and executors. The traces are in eXtensible Event Stream (XES) format⁵ [30]. XES is the de facto standard for process execution log expression and adopted by

---

⁴www.jboss.org/jbpm

⁵http://www.xes-standard.org/
several log analyzer tools such as the popular process mining framework, ProM⁶.

At diagnosis time, a BP execution social analysis component discovers and builds the necessary category of networks (configuration, machine, and social) based on process execution logs as well as BP models. This component is developed as a plugin of the ProM framework and represents the discovered networks using a well-known XML-based format for representing graphs, GraphML⁷. This format is supported by several graph drawing and analyzing tools such as yED Graph Editor⁸ and Eclipse Plugin Postfuse⁹. Therefore, such tools can also be used to visualize and analyze the discovered networks by the BP execution social analysis component.

Fig. 4 depicts examples of networks related to the electronic-patient-folder system that are generated using the BP execution social analysis component. For example, a network of machines is built to specify the backup relation between $m_1$ (scanner) and $m_2$ (three-function-printer) since both machines have the similar capacities (using subsumption). A configuration network of tasks is also constructed to express the interchange relation between $t_1$ (scan-documentation) and $t_2$ (enter-patient-details-manually). Both tasks produce similar output and their requirements do not overlap. Last and not least, a social network of persons is built to describe the peering relation between $p_2$ (cashier) and $p_3$ (financial-manager) since both persons have complementary capacities that are necessary to achieve $t_5$ (prepare-bill). This social network expresses also a substitution relation between $p_3$ (general-practitioner) and $p_4$ (emergency-physician) since both have the same capacity.

6. Conclusion

This paper discussed business processes from a social perspective. Different relations with a “social flavor” have been proposed such as interchange, substitution, delegation, and peering. They permitted to connect the components of a business process that are task, person, and machine together. A case study illustrated these components as well as these relations. For instance, tasks can be substituted when their requirements are not met. Persons delegate their tasks when unexpected changes occur in their schedules. Last but not least machines act as backups when their peers fail. The role of all these networks in conflict resolution is detailed in [26]. In term of future work, we would like to conduct some acceptance tests by end-users and to extend the analysis component of SUPER to handle cases like load balancing and resource management.

References


