A role-based collaborative process design
on crowdsourcing systems

ABSTRACT. Crowdsourcing is a collaborative business process model, in which tasks are carried out by a crowd. In crowdsourcing systems, there are two types of stakeholders namely, requesters who outsource tasks, and the crowd or contributors, performing those tasks. We consider a stakeholder as an actor or a standalone software component, evolving on a platform and having both mechanisms of interaction with its environment and business skills. A set of stakeholders interacting in a dynamic context for solving a problem, is a distributed collaborative system, and we term it crowdsourcing system. In such a system, the role concept is central, because each stakeholder must have a specific framework within which he collaborate. Traditionally, collaborative systems lose flexibility if their design is role-based, because only static role description mechanisms based on intuitive concepts are available. We propose in this paper, an improvement consisting of four things: (1) defining clearly what an outsourceable task or crowd task is, (2) specifying roles clearly and rigorously, while ensuring flexibility for collaboration, (3) providing role switching mechanisms and, (4) providing an abstract basis, for crowdsourcing system design and workflow monitoring and checking mechanisms, for potential activities, dynamically carried out by a system.

RÉSUMÉ. Le crowdsourcing est un modèle de processus métier collaboratif, dans lequel les tâches sont externalisées. Dans ces systèmes, on distingue deux types d’intervenants: les demandeurs qui externalisent des tâches et la foule ou contributeurs qui effectuent ces tâches. Nous considérons un intervenant à la fois comme un acteur et un composant logiciel autonome, évoluant sur une plateforme, disposant de mécanismes d’interaction avec son environnement et de compétences métiers. Un ensemble d’intervenants interagissant dans un contexte dynamique pour résoudre un problème, est un système collaboratif distribué, désigné par système de crowdsourcing. Dans un tel système, le concept de rôle est central, car chaque intervenant doit disposer d’un cadre spécifique dans lequel il collaborer. Traditionnellement, les systèmes collaboratifs perdent en flexibilité si leur conception est rôle-centrée, car seuls les mécanismes statiques de description de rôles, basés sur des concepts intuitifs sont disponibles. Nous proposons dans cet article, une amélioration consistant en quatre choses: (1) définir clairement ce qu’est une tâche externalisable, (2) spécifier les rôles clairement et rigoureusement, tout en assurant la flexibilité de la collaboration, (3) fournir des mécanismes de changement de rôle et, (4) fournir une base abstraite, pour la conception de systèmes de crowdsourcing et les mécanismes de monitoring et de vérification de workflows, pour les activités menées dynamiquement par un système.

KEYWORDS: Role-based approach, Dynamic workflow, Separation of concerns, SOD, Interface of role, Crowdsourcing Systems, Distributed collaborative systems, Guarded attribute grammar.

MOTS-CLÉS: Approche rôle-centrée, workflows dynamiques, séparation de préoccupations, SOD, interface de rôle, systèmes de crowdsourcing, systèmes collaboratifs distribués, GAG.
1. Introduction

Crowdsourcing concept, introduced by Jeff Howe [4], is a collaborative business process model, in which tasks are carried out by a crowd. Simply, an organization can request an online community for a voluntary accomplishment of a task, according to what both parties have a mutual benefit. In a crowdsourcing system, organizational objectives are top-down, while creative activity is bottom-up organized; we therefore say crowdsourcing is a mixed mode organized process [3]. Crowdsourcing processes are based on three pillars, namely: requester, crowd and crowd task [5]. 

Requester is a natural or legal person, who requests the power and wisdom of the crowd, for the performance of a given service. The requester, among other things, can encourage contributors by gratification or social motivation (public recognition), outsource tasks, check the compliance of results with predefined standards and finally ensure confidentiality on contributors data.

Crowd is a community of contributors, taking part in a crowdsourcing activity, whose characteristics are: diversity (variety of skills, spatiality,...), anonymity, importance, completeness or adequacy.

Crowd task is an activity in which the crowd participates. It can be a large-scale data collection, a co-creation task, or an innovation task. Characteristics of a crowd task are: modularity, complexity, solvability, automaticity, contributor-centric. Crowd skills orchestration, consists of distributing tasks to contributors, aggregating results and continuing the process based on the partial results obtained. According to task complexity, acceptance of the result and gratification granted criteria, there are mainly 3 strategies for orchestrating crowd services [6, 7]: market or strategy for large scale micro-tasking; contest or strategy for co-creativity, auction or strategy for innovation.

A crowdsourcing system is an environment, where a crowdsourcing process is deployed; it integrates practices of business process modeling, inspired by service-oriented architecture (SOA) moreover, service calls may not be hierarchically organized; it establishes connection between crowd, individual actors and machines. It allows services composition by assembling a suite of small specialized independent services, offering respectively modular decomposition basis for services, an architecture organized around business skills, focusing on product but not the project. Communication here is either Restful, SOAP like or by messages through channels; governance and data management are decentralized. Defining properly a stakeholder requires specifying two things: (i) its functional goal or business skills, i.e interface (services he requires and services he provides) and business rules (functionalities targeted by the stakeholder); (ii) its infrastructure mechanisms or intrinsic skills i.e storage for persistence, communication mechanisms, sensing tools, etc [15]. In this work our interest is crowdsourcing processes design for large scale micro-tasking. We consider a crowdsourcing system, as a set of independent stakeholders providing precise services. Stakeholder infrastructure is thus perceived as an actor, while its functional goal is a role. In fact, in such a system, each actor can be assigned several roles and several actors could play one instance of role. Actors may not have same intrinsic skills, and several occurrences of a role could exist at the same time in the system. Considering all role in a given context of a system, we get a formal reasoning basis for business goals of the whole system. As crowdsourcing system use case, consider a city participatory management case [16], with processes depict on figure 2, where citizens (BOB, JANE, ALICE, etc.) via the urban information system (URBANIS), provide information on the state of city roads and determine which ones to maintain as a priority. Cleaning is done (by CLEANING_CO) on the targeted roads, while municipal executive
(MunicipalExecutive) contracts with a company (Road_CO) in order to carry out the work.

Figure 1 – A crowdsourced road maintenance activity

Role concept is central in any collaborative system, as each actor must have a clear framework within which he collaborate. A role specifies both what the system expects from actors, but also what actors expects from the system; thus avoiding that an actor be overwhelmed by information (or tasks) not necessary. Traditionally, collaborative systems lose flexibility as soon as their design is role-based, because only static role description mechanisms, based on intuitive concepts, are available [2], and certainly, a dynamic context of collaboration, further complicates a design of such systems, since entities involved, evolve over time in number and in skills. An improvement could be to provide four things: (1) define clearly what an outsourceable task or crowd task is, (2) specify roles clearly and rigorously, while ensuring flexibility for collaboration, (3) provide role switching mechanisms and, (4) provide an abstract basis, for crowdsourcing system design and workflow monitoring and checking mechanisms. This design approach implies a rigorous definition in advance, of the set of roles and the relationships between them, necessary to describe the functioning of a target domain. Role being a particular concern in the system. In fact, a role-based design approach is similar to a separation of concern technique [1, 11], applied to business process design. It is implemented by specifying activities defining process’s steps, as well as flows describing coordination of these activities, as it may be done with BPMN orchestration or UML collaboration or activities diagrams.

2. Role & role interface modeling and basic mechanisms

Common definitions and terminologies - a role type is the perception one actor has of another actor [13]. A role type is specified uniquely, and an actor plays at a given time a role specified by a role type. A role is therefore defined as one of the instances of a given role type, played by an actor. It may not always be easy to differentiate between a role and an actor. For instance, consider the Mayor and CSOfficer entities, as parts
of collaboration within a process of civil status certificate issuing at appendix A1. Being able to distinguish between these entities, which is role type (or simply role) and which is actor, can be rather complex. The sharp distinction between an actor and a role type is based on both concepts of foundation and semantic rigidity [14]. An entity is considered to be founded, if its specification implies a dependency or relationship with another entity. Rather it is semantically rigid, if its identity depends on certain characteristics, and can not exist without them. Thus, it will be said that MAYOR is unfounded and semantically rigid (and therefore MAYOR is an actor), whereas CSOFFICER is founded and semantically non-rigid (CSOFFICER is a role); in other words, a MAYOR plays the CSOFFICER role. Hence an actor is unfounded and semantically rigid, while a role is founded and semantically non-rigid.

Role and crowdsourcing grammatical tasking model - In this work we will consider, role type and role as equivalents, simply called role. A role specifies the set of actions a given actor can take in the system.

Definition 2.1 A role \( r \) is given by a couple \((G, R)\) where \( G \) is a guarded attribute grammar (GAG)\(^1\) specifying role \( r \) business skills, and \( R \) it’s associated interface of role.

Business skills are expressed as grammar production rules (or business rules), describing job decomposition, in the form

\[
s_0 \rightarrow T_0 \cdots T_m s_1 \cdots s_n \bar{s}_0 \cdots \bar{s}_p
\]  

(1)

where \( s_0 \) is a defined service, \( s_1 \cdots s_n \) are used services and \( T_0 \cdots T_m \) are actor’s intrinsic skills, finally \( \bar{s}_0 \cdots \bar{s}_p \) are crowd tasks. According to this rule, to supply service \( s_0 \), services \( s_1 \cdots s_n \) and \( \bar{s}_0 \cdots \bar{s}_p \) must be achieved and skills \( T_0 \cdots T_m \) are needed by actor playing that role. Within the role and considering business rules, services defined but not used, are called provided services, those used but not defined are required services, lastly services defined and used are internal services. Crowd tasks are defined by business rules like \( \bar{s}_i \rightarrow T_0 \cdots T_m \) (\( 0 \leq i \leq p \)) i.e they are type of services requiring only actors intrinsic skills, to be carried out and so, they are autonomous services. Consider a business rule \( s_0 \rightarrow T_0 \cdots T_m \) one may ask the difference between services \( s_0 \) and \( \bar{s}_0 \). In fact \( s_0 \) is a potentially outsourceable task, it can be defined and used in the same role (or used by another role); rather \( \bar{s}_0 \) is an outsourced task, so it is only defined in an autonomous role, term crowd role, and only used by a requester role. By convention, a crowd role only define outsourced tasks, and we say a crowd role supply only crowd services. For instance, if we reconsider the crowdsourcing system described on figure 2, Any SENSOR, can snap an object, in a predefined context (state of the object) and (geographic) location (\( \text{clSnap} \)) on the one hand or quite simply according to the circumstances of the moment (\( \text{snap} \)), following business skills expressed by business rules in (2).

\[
\text{clSnap} \rightarrow \text{CONTEXT LOCATION SNAP} \\
\text{snap} \rightarrow \text{SNAP}
\]  

(2)

---

1. As the goal is to describe processes, we are not yet interested with attributes in guarded attribute grammars.
A CITIZEN according to business skills in rule (3), can tag a picture and assess object, in both cases by inputting some data.

\[
\begin{align*}
tagPicture & \rightarrow \text{INPUT} \\
\text{assessObject} & \rightarrow \text{INPUT}
\end{align*}
\] (3)

Lastly according to rule (4), an IS i.e information system is responsible of data collection, selecting targeted road for maintenance, and sending some alerts (rules are not yet defined).

\[
\begin{align*}
\text{selectRoad} & \rightarrow \text{clSnap} \\
\text{alert} & \rightarrow \varepsilon
\end{align*}
\] (4)

**Interface of role** - an *interface of role* [8, 9] is some abstraction of GAG G associated to a role, specifying what services are provided, which external services are required to carry them out and an over-approximation of dependencies between required and provided services termed *potential dependencies*. Interface disregards internal tasks. See [9] for interfaces derivation from a GAG.

**Definition 2.2** Let \( \Omega \) a fixed set of services. An interface \( (^*R, R, R^*) \) consist of a finite binary relation \( R \subseteq \Omega \times \Omega \) of disjoint sub-sets \( ^*R \) and \( R^* \) from \( \Omega \), such that \( ^*R = R^{-1}(\Omega) = \{ A \in \Omega \mid \exists B \in \Omega \ (A, B) \in R \} \) and \( R^* \supseteq R(\Omega) = \{ B \in \Omega \mid A \in \Omega \ (A, B) \in R \} \).

Set \( R^* \) represents services provided (or defined) by the interface and \( ^*R \) is the set of required (or used) services. Relation \( (A, B) \in R \) indicates that service \( B \) potentially depends upon service \( A \). Thus \( A \in R^* \setminus R(\Omega) \) is a service provided by the interface that requires no external services. An interface is closed (or autonomous) if relation \( R \) (and therefore also \( ^*R \)) is empty. Thus, a closed interface is given by all services it (autonomously) provides.

\[
\begin{align*}
^*\text{IS} & = \{\text{clSnap}, \text{snap}, \text{tagPicture}\} \\
\text{IS} & = \{\text{clSnap, selectRoad}, \text{snap, selectRoad}, \text{tagPicture, selectRoad}, (_-, \text{alert})\} \\
\text{IS}^* & = \{\text{selectRoad, alert}\}
\end{align*}
\] (5)

An interface provide several basic operations [8, 9] as:

**Restriction (\( \upharpoonright \))**: Let \( O \subseteq \Omega \) a subset of services. A *restriction* of interface \( R \) to \( O \) denoted \( R \upharpoonright O \) is given by \( R \upharpoonright O = \{ (A, B) \in R \mid B \in O \} \). With \( (R \upharpoonright O)^* = O \cap R^* \) and \( ^* (R \upharpoonright O) = R^{-1}(O \cap R^*) \) respectively. Considering a role \( r \) interfaced by \( R \), and a subset of services \( O \), the restriction operation reconfigures role \( r \) so that, only its provided services elements of \( O \) are enabled; useful when just some skills of a multi-skilled role are needed.

**Cascade composition test (\( \Join \))**: Let two roles \( R_1 \) and \( R_2 \): cascade composition of those roles holds iff \( R_1^* \cap ^* R_2 = \emptyset \wedge ^* R_1 \cap R_2^* \neq \emptyset \). We denote \( R_1 \Join R_2 \) their cascade product test (or \( R_2 \Join R_1 \) since this operation is commutative). When cascade composition holds, the resulting role required services set is \( ^* (R_1 \setminus R_2^*) \cup ^* R_2 \), and provided services set is \( R_1^* \cup R_2^* \).
Direct Product (×) Let \( R_1 \) and \( R_2 \) two composable role interfaces. If \( R_1^* \cap \star R_2 = \emptyset \) and \( R_2^* \cap \star R_1 = \emptyset \), the composition is the product (direct) of \( R_1 \) and \( R_2 \), denoted by \( R_1 \times R_2 \). Note that \( R_1 \times R_2 = R_1 \cup R_1 \) and so \( \star (R_1 \times R_2) = \star R_1 \cup \star R_2 \) and \( (R_1 \times R_2)^* = R_1^* \cup R_2^* \).

3. Collaborations and collaboration schemes

In a targeted domain, a context of collaboration is the set of roles available instantly for the realization of some activities. A context of collaboration is dynamic, i.e., roles involved vary over time. For instance, we will consider the context on Figure 3 Appendix A4, as a running collaborative context, for the next parts of this work. We talk of collaboration between two roles, when there is a service dependency between them. This dependency can be direct, in which case it is a direct collaboration; likewise, it can be indirect being an indirect collaboration.

Definition 3.1 Two roles \( r_1 = (G_1, R_1) \) and \( r_2 = (G_2, R_2) \) are in a direct collaboration iff \( R_1 \times R_2 \) holds. We denote \( \star (R_1 \cap R_2^*, r_1) \) that collaboration, labeled by \( \star R_1 \cap R_2^* \), the set of services for which \( r_1 \) and \( r_2 \) collaborate; \( r_1 \) being the services requester, while \( r_2 \) is the provider of those services. Thus \( r_1 \) and \( r_2 \) will be in an indirect collaboration iff \( \exists r_k = (G_k, R_k) \) such that \( R_1 \times R_k \times R_2 \) holds.

Potential direct collaborations of a role - a potential direct collaborations of a role, is a graph showing all potential services providers for that role in a collaborative context. Let \( r_0 \) be a given role; algorithm 1 (Appendix A2), determines potential direct collaborations (or potential dependencies) of \( r_0 \), in a context \( R \), in which \( r_0 \) is member and such that \( \forall r_i \in R \), \( r_i = (G_i, R_i) \). Applying algorithm 1 on context (figure 3), will result to graph \( C = \{(x, r_0, r_3) , (x, r_4, r_3), (t, r_2, r_3), (w, r_1, r_3)\} \).

Collaboration schemes - from any context \( R \), a collaboration scheme or induced potential dependencies graph is obtained by grouping step by step, all the potential dependencies of the various roles in the context, as implemented by the algorithm 2. Applying that algorithm on the previous context \( R \), may leads to collaboration scheme \( C = \{(a, r_5, r_0) , (m, r_6, r_4), (y, r_0, r_2), (z, r_1, r_2), (w, r_1, r_3), (x, r_4, r_3), (x, r_0, r_3), (t, r_2, r_3)\} \).

Potential workflow of a service - the induced workflow of a service \( s_0 \), describes how this service will be issued; it is implemented as a dependency subgraph, derived from the induced potential dependencies graph (\( iPDG \)) of the role providing service \( s_0 \). Consider a predicate \( depend(l, s_0) \) with \( l, s_0 \in \Omega \), which returns true if service \( s_0 \) depends on the service \( l \) and false otherwise. We define function \( dependOn(s_0, R) = \{(L, r) | L \subset \Omega^*, r \in R \ and \ \forall l \in L \ depend(l, s_0) \} \) which seeks in context \( R \), all the roles involved in the process of providing service \( s_0 \), as well as associated required services, and returns a list of couples made up of a required services set \( L \) and the role \( r \) requesting these services. We also let \( iPDG(R, \emptyset) \mid L_k = \{(l_0, r_i, r_j) \mid \forall (L_k, r_k) \in dependOn(s_0, R), \ l_0 \in L_k \ and \ r_j = r_k \} \) be a filtering made on the induced potential dependencies graph, concerning collaborations \( (l_0, r_i, r_j) \) such as for any couple \( (L_k, r_k) \in dependOn(s_0, R) \), \( r_k \) being requester of service \( l_0 \) with \( (l_0 \in L_k \ and \ r_j = r_k) \). Algorithm 3 (Appendix A2), generates the potential workflow of a given service \( s_0 \), from
A role-based collaborative process design on crowdsourcing systems

For instance, the potential workflow of the service $u$, obtained from that algorithm in the context of previous figure 3, is given by $workflow(\{u\}, R) = \{(x, r_4, r_3), (x, r_0, r_3), (t, r_2, r_3), (m, r_6, r_4), (a, r_5, r_0), (y, r_0, r_2), (z, r_1, r_2)\}$

**Factorizing a workflow** - a collaboration scheme may include several alternatives in supplying the same service, as the one of figure 4(a)-Appendix A3, where service $x$ is provided by roles $r_0$ and $r_4$ respectively. In a given context factorizing, it is able to transform cases such as for a role $r_3$, requesting service $x$, so that we have service $x$ potential suppliers list; as it is shown on figure 4(b)-Appendix A3.

**Definition 3.2** An $F-$ collaboration is a triplet $(\star R_0 \cap R^*_i \{r_1, \ldots r_k\}, r_0)$, where $r_1, \ldots r_k$ are potential providers of services elements of set $(\star R_0 \cap R^*_i)$ and $r_0$ is the requester for those services (with $(\star R_0 \cap R^*_i = \cdots = \star R_0 \cap R^*_i)$, for some $i$ and $j$).

A factorized collaboration scheme is then a potential dependency graph, possibly consisting of collaborations (i.e $(\star R_0 \cap R^*_1, r_1, r_0)$, and $F-$ collaboration $(\star R_0 \cap R^*_k \{r_1, \ldots r_k\}, r_0)$) if necessary. Let $P(C) \subseteq \mathcal{P}(C)$ a subset of parts of set $C$, where elements are grouped subsets of all identically labeled collaborations. Algorithm 4 transforms a $C$ collaborations scheme to a $C'$ factorized collaborations schemes.

4. Activity in collaborative context

**Definition 4.1** An activity for a given service $s_0$, in a context $R$, is a couple denoted $activity_{s_0} = (s_0, workflow(\{s_0\}, R))$, and is the process of supplying service $s_0$, described by $workflow(\{s_0\}, R)$.

An activity can have several occurrences of the same role (indifferently supplier or requester). If two roles $r_0$ and $r_1$ respectively, provide the same service $s_0$ within an activity, then they do not necessarily use the same required services i.e. $rPDG(r_0, R) \neq rPDG(r_1, R)$.

**Proposition 4.1** Two activities $activity_{s_0}$ and $activity_{s_1}$ are equivalent, and we denote by $activity_{s_0} \equiv activity_{s_1}$, iff $s_0 = s_1$ and $ workflow(\{s_0\}, R) \equiv workflow(\{s_1\}, R)$ i.e they deliver the same service in context $R$, with $s_0, s_1 \in \Omega$.

**Proof 1** Consider $R_1 \ldots R_m$ as pairwise composable role interfaces involved in a given workflow$(\{s_0\}, R)$ and $R'_1 \ldots R'_n$ those of workflow$(\{s_1\}, R)$ respectively, with $m \neq n$. Let $R = \times_{i=1}^m R_i$ and $R' = \times_{i=1}^n R_i$ their respective cascade compositions. By proposition 4.5 in [8, 9], those compositions are associative. As by hypothesis those workflows render same services, we have $s_0 = s_1$ and $s_0 \in R^* \cap R'^*$, two cases can be distinguished: whether $m > n$ and then $R' \subset R$, we say $R'$ realizes service $s_0$ with less business rules than $R$; or $m < n$ so $R \subset R'$ and as $R'$, $R$ realizes service $s_0$ with less business rules.

**Atomicity of activities** - an activity for a given service $s_0$, is said to be atomic [1], if it has only one occurrence of role supplier for each required service in that activity. For example, the atomic forms of previous activity $activity_u$ are respectively: $activity_{u_0} = (u, [(x, r_0, r_3), (t, r_2, r_3), (a, r_5, r_0), (y, r_0, r_2), (z, r_1, r_2)])$, $activity_{u_1} = (u, [(x, r_4, r_3), (t, r_2, r_3), (m, r_6, r_4), (y, r_0, r_2), (z, r_1, r_2)])$
Definition 4.2 An activity \( a \) is atomic, iff for all \( (s_0, r_i, r_j) \) in \( \text{workflow}(s_0, R) \), \( r_i = r_j \).

Activity decomposition - An activity can be progressively fragmented into a set of atomic activities. The principle of decomposition, is based on roles (concern), and states that, as long as there are several occurrences of the same role \( r \) in an activity, this activity is broken down into new activities containing a single role occurrence \( r \). This principle is repeated until all activities obtained are atomic [1]. For this, associated workflow must be factorized; if at the end of this, \( F - \text{collaborations} \) exist, then activity is decomposable, according to principles of algorithm 5 (Appendix A2), and figure 5 (Appendix A4).

Proposition 4.2 Consider activity \( (s_0, workflow_0(s_0, R)) \) and activity \( (s_0, workflow_1(s_0, R)) \) two activities, where activity \( s_0 = activity'_0 \). activity \( s_0 \) is decomposable to activity \( s_0' \) if and only if \( workflow_0(s_0, R) \neq workflow_1(s_0, R) \) and factorize(\( workflow_0(s_0, R) \)) = \( workflow_1(s_0, R) \)

Proof 2 As the two activities are equivalents by hypothesis, the demonstration is equivalent to show that \( a \) in a workflow, several collaborations for given service, is equivalent to \( F - \text{collaborations} \) on the same service; and this is done by definition 3.2.

Activity Realizability - refers to the possibility of carrying out an activity, in a finite number of stages, and rendering provided service. This assumes that all necessary roles are available instantly; we say it is a favorable context. Termination of an activity, is conditioned by the fact that its workflow must contain autonomous roles as triggers. Algorithm 6 (Appendix A2), describes feasibility principles for a workflow, by applying a pattern matching mechanism. A workflow is realizable if that algorithm returns \( True \) and its required services queue(Serv) is empty. In case that \( False \) is returned, required service queue contains a list of services still to be provided, for completing activity.

Remark. — An activity \( a \) is said to be quasi realizable, if at least one of its atomic forms obtained by decomposition, ends; i.e there is a \( C \in \text{decomp}(\text{workflow}(s_0, R), \emptyset) \) such that \( \text{realizable}(\{s_0\}, C) = (True, \emptyset) \). Similarly the activity \( a \) is said realizable, if all atomic forms end; i.e. whatever \( C \in \text{decomp}(\text{workflow}(s_0, R), \emptyset) \), \( \text{realizable}(\{s_0\}, C) = (True, \emptyset) \).

5. Actor of a distributed collaborative system

Definition 5.1 An actor \( a \) is given by a couple \( (R, C) \), where \( R \) is the set of potential roles that actor can play, and \( C \) is the set of constraints on those potential roles.

Let \( a = (R, C) \) an actor, \( r_i \) and \( r_j \) two roles; association between actor \( a \) and roles \( r_i \) and \( r_j \) is materialized by \( r_i, r_j \in R \). We will say for instance that \( r_i, r_j \) are actor’s potential roles. Four constraint values can be defined on actor’s potential roles [13], namely: \text{Dcr} or nothing for no constraint; \text{Imp} \( r_i, r_j \) indicating that if actor \( a \) plays role \( r_i \), then he must also play role \( r_j \); \text{Eqv} \( r_i, r_j \) in case both \text{Imp} \( r_i, r_j \) and \text{Imp} \( r_j, r_i \) holds; \text{Phb} \( r_i, r_j \) and so, actor \( a \) playing role \( r_i \), cannot play role \( r_j \).

An actor can play one or more roles, within an activity, or in several parallel activities; then three cases of "play" relationship can be distinguished: Playing several roles in an
activity an actor can play more than one role within an activity; provided that those roles do not provide same services. An actor $a_\tau$ playing several roles $r_0$ and $r_1$ respectively, in activity $s_{0_\tau} = (s_0, \ldots, (B, r_1, r_2), (C, r_2, r_0), \ldots)$, see figure 6(a), is multi-skilled in that activity. Therefore, the different roles of $a_\tau$ can be pooled into a single macro-role $r'_0$ whose interface is $R_0 \times R_1$. So activity $s_{0_\tau}$ can undergo a transformation to become $s'_{0_\tau} = (s_0, \ldots, (B, r'_0, r_2), (C, r_2, r_0), \ldots)$, as illustrated in the figure 6(b). Competing activities an actor can contribute in several activities at the same time, either by playing the same role each time (see figure 6 (c)), or by having different roles. In all these cases, each of these activities is implemented individually as in case 1 above. Crowd role played by several actors within an activity, several instances of actors with same intrinsic skills, can play the same occurrence of a role; in a workflow, if role $r$ is played by several actors instances (figure 6 (d)), then any collaboration with $r$ will be on a crowd task, and $r$ is a crowd role. A pattern matching implementation of play relation between an actor and his potential roles, is given by equation 9 on Appendix A3.

6. Conclusion

This work focused on a role-based design approach, in a micro-tasking crowdsourcing system context i.e those dynamic system where every actor can be assigned several roles, can come in and go out of the system as he pleases and, several instances of actors can play the same role occurrence at a given moment, in the system. In our approach, we made a distinction between actor i.e infrastructure of a stakeholder and role i.e his business skills. Considering all skills in the system at a moment, procure an abstract basis for reasoning about business goals of the whole system. We defined a rigorous specification method for roles some basic operations, namely collaboration and roles re-configuration. We also provide decomposition mechanisms and workflow monitoring and checking tools, role switching and workflow simplification mechanisms, the play relationships between an actor and his assigned roles, and the constraints between roles. This work is a prelude to a role-based design approach, for distributed collaborative systems. An immediate continuation is to reconsider a more complex system, as co-creative and innovative crowdsourcing systems, i.e a context where there are no pre-established rules, clarify conception of those type of processes, and describe interactions between actors.

7. References


Appendix

A1. E-administration collaboration use case

Consider an E-administration application for issuing civil status certificates, described on figure 2. This scheme shows collaborations between roles DECLARER, SECRETARY, CSOFFICER and JUDICIARY_AUTHORITIE; roles played respectively by actors BOB, AGENT, (MAYOR, DIPLOMAT) and PROSECUTOR as stakeholders in an issuance civil status certificate (birth, marriage, death) activity.

Any DECLARER, can trigger the process of establishing a civil status certificate, by a declaration which can be either normal (dcl) or special (sDcl), following business skills expressed by business rules in (6).

\[
\begin{align*}
\text{dcl} & \rightarrow \text{INPUT withnessing} \\
& \quad \text{INPUT hospital} \\
& \quad \text{INPUT reading} \\
\text{sDcl} & \rightarrow \text{INPUT}
\end{align*}
\] (6)

A SECRETARY according to business skills in rule (7), is responsible of declarations correction and transcription in related registers, whether these declarations are normal (normAscr) or special (spAscr).

\[
\begin{align*}
\text{normAscr} & \rightarrow \text{TRANSCRIPTION dcl} \\
\text{spAscr} & \rightarrow \text{TRANSCRIPTION sDcl judgment} \\
\text{reading} & \rightarrow \text{CORRECT dcl}
\end{align*}
\] (7)

Lastly according to rule (8), a CSOFFICER is responsible of issuing civil status certificates and certain checks. Checks rules are not yet expressed and may be done later.

2. According to Cameroonian laws
A2. Related algorithms

Algorithm 1: Role potential dependencies graph (rPDG) calculus

1. **input**: \( r_0 = (G_0, R_0) \), \( R \)
2. **output**: \( C \) (set of potential collaborations of role \( r_0 \)).

\[
C \leftarrow \emptyset
\]

\[
rPDG(r_0, R) =
\]

\[
\text{forall } r_i \in R \\
\quad \text{if } R_0 \sqsubseteq \sqsubseteq \sqsubseteq R_i \text{ then } C \cup \{ (\star (R_0 \cap R_i^*), r_i, r_0) \}
\]

Algorithm 2: Context \( R \) induced potential dependencies graph (iPDG)

1. **input**: \( R \)
   - \( R' \) (roles whose rPDG have already been determined, initially empty).
2. **output**: \( C \)

\[
R' \leftarrow \emptyset
\]

\[
iPDG(R, R') =
\]

\[
\text{if } (r_i \in R) \text{ and } |R| \neq 0 \text{ then}
\quad rPDG(r_i, R \cup \{R' \cup \{r_i\}\}) \cup iPDG(R \setminus \{r_i\}, R')
\]

\[
\text{else } iPDG(R \setminus \{r_i\}, R')
\]

Algorithm 3: Determining a potential workflow for a service

1. **Inputs**: \( Serv = \emptyset \cup \{s_0\}, R \)
2. **output**: \( C \) (a set of potential collaborations needed to provide the service \( s_0 \)).

\[
\text{workflow(Serv, R)} =
\]

\[
\text{if } s_i \text{ in Serv then } - i \in \{1, \ldots, |Serv|\}
\quad ns = Serv \setminus \{s_i\} \cup \{s \mid (s, r) \in dependOn(s_i, R)\}
\]

\[
\{iPDG(R, \emptyset) \uparrow L_k \} \cup \text{workflow}(ns, R, iPDG(R, \emptyset))
\]

\[
\text{where } (L_k, r_k) \in dependOn(s_0, R)
\]
A role-based collaborative process design
on crowdsourcing systems

Algorithm 4: Factorizing a service

1. input: C
2. output: C'
3. C' ← ∅
4. factorize(C) =
5. forall c in P(C)
6. if |c| == 1 then //c is like {(*R₀ ∩ Rₖ, r₁, r₀)}
7. C' ∪ c
8. else if |c| > 1 then //c is like {(*R₀ ∩ Rₖ, r₁, r₀), ..., (*R₀ ∩ Rₖ, rᵢ, r₀)}
9. C' ∪ {(*R₀ ∩ Rₖ, {r₁, ..., rᵢ}, r₀) | rᵢ≤i≤|c|} ∈ provider {*R₁ ∩ Rₖ|c|, c} and r₀ == requester(c)
10. function provider {*R₁ ∩ Rₖ|c|, c} returns the list of service providers, labeled by elements of set *R₁ ∩ Rₖ|c| in a c collaboration set, while requester(c) checks if r₀ is the requester in each case of collaboration.

Algorithm 5: Atomic decomposition of an activity’s workflow

1. input: C' //factorized workflow of the activity.
2. output: C //set of potential atomic workflows.
3. C ← ∅
4. decompose(C', C) =
5. forall c in C'
6. if |provider(c)| == 1 then //c is like {*R₀ ∩ Rₖ, {r₀}}
7. decompose(C' \ {c}, insert (c, C))
8. else if |provider(c)| > 1 then //c is like {*R₀ ∩ Rₖ, {r₁, ..., rᵢ}, r₀}
9. decompose(C' \ {c}, mdup (c, C))

insert (c, C), insert collaboration c, in the different atomic workflows of C, for which c is necessary.
mdup (({*R₀ ∩ Rₖ, {r₁, ..., rᵢ}, r₀}), C) add in each atomic workflow of C, collaborations {*R₀ ∩ Rₖ, rᵢ, r₀} with r ∈ {r₁, ..., rᵢ}, as long as these collaborations are useful, for the realization of the service associated with that workflow.

Algorithm 6: Checking realizability of activityᵦ₀

1. input: Serv //set of required services
2. output: (Bool, Serv)
3. realizable(∅, ∅) = (True, ∅) //the workflow is realizable
4. realizable(Serv, ∅) = (False, Serv) //not realizable, Serv are required services
5. realizable(Serv, c ∈ C)
6. | Serv ∩ label(c) == ∅ = realizable(Serv ∪ req, C \ {c})
7. | Serv ∩ label(c) ≠ ∅ = realizable(Serv' ∪ req, C \ {c})
8. where Serv' = Serv \ {x ∈ Serv ∧ x ∉ label(c)}
9. req = dependOn(label(c), provider(c))

For a given collaboration c = {*R₁ ∩ Rₖ|c|, rᵢ, r₁}, realizable returns true if that collaboration is realizable, and provider(c) returns rᵢ providing those services.
A3. Implementation of the "play" relationship

Consider three primitives $\text{play} \left(a_\tau, \{r_i\}_{1 \leq i \leq |R_\tau|}, \text{activity}_{s_0}\right)_{cstr_k} \{1 \leq k \leq N\}$ indicating that $a_\tau$ potentially can play roles $\{r_i\}_{1 \leq i \leq |R_\tau|}$ in activity $\text{activity}_{s_0}$, with constraints $\{cstr_k\}_{1 \leq k \leq N}$; $\text{play} \left(a_\tau, r_i, \text{activity}_{s_0}\right)_{cstr_k}$ to express that $a_\tau$ potentially can play the roles $r_i$ in activity $\text{activity}_{s_0}$, according to the constraint $cstr_k$, with

$$\text{play} \left(a_\tau, \{r_i\}_{1 \leq i \leq |R_\tau|}, \text{activity}_{s_0}\right)_{cstr_k} \{1 \leq k \leq N\} = \bigcup_{1 \leq i \leq |R_\tau|} \text{play} \left(a_\tau, r_i, \text{activity}_{s_0}\right)_{cstr_k}$$

and finally $\text{play} \left(a_\tau, R_i, \text{activity}_{s_0}\right)$ expressing that $a_\tau$ actually plays the role $r_i$ whose interface is $R_i$, in activity $\text{activity}_{s_0}$. Possible implementations of the "play" relationship while taking account of constraints on roles, are described by equations 9 given below:

$$\text{play} \left(a_\tau, r_i, \text{activity}_{s_0}\right)_{Dcr} = \text{play} \left(a_\tau, R_i, \text{activity}_{s_0}\right)$$
$$\text{play} \left(a_\tau, r_i, \text{activity}_{s_0}\right)_{Imp(r_i, r_j)} = \text{play} \left(a_\tau, R_i \times R_j, \text{activity}_{s_0}\right)$$
$$\text{play} \left(a_\tau, r_i, \text{activity}_{s_0}\right)_{Eqv(r_i, r_j)} = \text{play} \left(a_\tau, R_i \times R_j, \text{activity}_{s_0}\right)$$
$$\text{play} \left(a_\tau, r_i, \text{activity}_{s_0}\right)_{Phb(r_i, r_j)} = \text{play} \left(a_\tau, R_i, \text{activity}_{s_0}\right) \text{ or play} \left(a_\tau, R_j \times R_i, \text{activity}_{s_0}\right)$$
$$\text{play} \left(a_\tau, r_i, \text{activity}_{s_0}\right)_{Phb(r_i, r_j)} = \text{play} \left(a_\tau, R_i, \text{activity}_{s_0}\right) \text{ and not play} \left(a_\tau, R_j, \text{activity}_{s_0}\right)$$

A4. Figures

Figure 3 – Running collaboration context of a system

Figure 4 – Factorizing a workflow - (a) a collaboration scheme, (b) the factorized collaboration scheme equivalent to the one on (a).
(a) Process of issuing a civil status certificate by the mayor

(b) Process for the issuance of a civil status certificate by a diplomat

Figure 5 – Decomposition of the previous activity on figure 2, into two sub-activities, $csDelivrance0$ and $csDelivrance1$

(a) Actor $a_\tau$ playing two roles

(b) Workflow simplified by the direct product of $R_0$ and $R_1$

(c) An actor involved in two parallel activities

(d) Several actors playing same role $r_2$ in an activity

Figure 6 – relation play scenarios